UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática y Matemáticas

TRABAJO FIN DE GRADO

IMPLEMENTACIÓN DE UNA REJILLA PROBABILÍSTICA DE OCUPACIÓN PARA LA CONDUCCIÓN AUTÓNOMA

Jorge Arellano Subías
Tutor: Jorge Villagra
Ponente: José Dorronsoro

Enero 2019

IMPLEMENTACIÓN DE UNA REJILLA PROBABILÍSTICA DE OCUPACIÓN PARA LA CONDUCCIÓN AUTÓNOMA

AUTOR: Jorge Arellano Subías
TUTOR: Jorge Villagra
PONENTE: José Dorronsoro

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Enero de 2018

Resumen

Este trabajo de fin de grado ha sido llevado a cabo en el CAR, Centro de Automática y Robótica. Se ha desarrollado un software para la detección y estimación de velocidades de obstáculos, a partir de datos obtenidos de un láser de cuatro capas y un GPS de alta precisión. Combinando los datos del instante presente con las predicciones obtenidas en instantes anteriores, se ha logrado estimar, con una gran precisión, la ocupación del espacio visible por el láser. Además, se estiman algunos espacios oclusos o fuera de rango, utilizando información recogida anteriormente. También se han desarrollado algoritmos novedosos para agrupar esta ocupación y lograr modelar objetos y sus velocidades. En conclusión: este software es capaz de mapear el espacio localizando obstáculos y sus velocidades.

En esta memoria, se recoge todo el proceso, describiendo tanto el diseño de los algoritmos como el desarrollo de éstos al detalle. Se comenta el estado del arte de la conducción autónoma. También se muestran algunos resultados y se proponen algunos trabajos futuros para continuar esta línea de investigación.

Palabras clave

Paper, visión, filtro de partículas, tiempo real, paralelizable, clustering, debugueo, rejilla, nube de puntos, LIDAR, falso impacto, resolución angular, dataset, pitch.

Abstract

This final degree project has been carried out in CAR (*Centro de Automática y Robótica*). A software for obstacle and detection and velocity estimation has been developed, using data provided by a four-layer laser and a high-precision GPS. Combining information of the present instant and historic information, the occupancy of the visible zones has been estimated with huge precision. Also, occluded and out of range areas are modelled using information gathered previously. Some new algorithms have been developed in order to group this occupancy and detect objects and calculate their velocities. In conclusion: this software is able to map the environment localizing obstacles and their velocities.

In this report, the entire process is documented, describing both the design of the algorithms and the development of these in detail. The state of the art of autonomous driving is also discussed. Some results are shown and some future works are proposed to continue this line of research.

Keywords

Paper, vision, particle filter, real time, parallelizable, clustering, debug, grid, point cloud, LIDAR, false impact, angular resolution, dataset, pitch.

Agradecimientos

Muchas personas me han ayudado para llevar a cabo este trabajo. En primer lugar, agradecer a Jorge Villagra Serrano la oportunidad de trabajar en un centro de investigación con tanto renombre, permitiéndome conocer a gente maravillosa, usar unas instalaciones excelentes, y descubrir cómo funciona el mundo de la investigación. Una mención especial también a Antonio Artuñedo y Jorge Godoy, compañeros del despacho, que me ayudaron múltiples veces no sólo a grabar datasets y procesarlos, si no a sentirme como en casa desde el minuto uno. En otro orden de magnitud, agradecer a Idoia Alarcón por darme a conocer estas becas, a Jose Dorronsoro por ofrecerse a ser mi ponente en la universidad, y por supuesto a mis padres Eugenio Arellano y Rosa Subías por apoyarme siempre en lo que hago.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

1 Introducción

1.1 Motivación

Esta memoria de TFG recogerá las investigaciones y avances llevados a cabo durante la estancia en el CSIC asociado a la UPM en Arganda. En concreto, se ha desarrollado bajo un proyecto más global llamado Autopía [1], enfocado al mundo de la conducción autónoma, un campo en pleno desarrollo y con mucho futuro por delante. Allí, cuentan con un coche completamente automatizado equipado con el software y hardware necesarios para que esto sea posible. Con él, han participado en varias competiciones y exhibiciones a nivel nacional y mundial, incluyendo demostraciones de planificación de rutas, detección dinámica de obstáculos, y conducción sincronizada entre varios automóviles autónomos.

Jorge Villagra, es el jefe del proyecto Autopía y tutor de este TFG: creación de una rejilla probabilística de ocupación. Este trabajo combina Matemáticas (modelización y probabilidad) con Ingeniería Informática (algoritmia y desarrollo de software). Así, a partir de algunos papers[1] recogidos en Referencias, se comienza a desarrollar este trabajo perteneciente a la rama Visión[2].

Este proyecto ha sido y será muy relevante para Autopía. En la conducción autónoma, es muy importante modelar la ocupación del entorno de una manera lo más precisa posible. No solo eso, sino que habrá que hacerlo en tiempo real, pues el coche tomará decisiones de ruta y velocidad en función de los datos que genere el software a desarrollar. Además, no solo interesa la ocupación del espacio, sino también diferenciar los objetos en él y sus velocidades. Para lograr todo esto, se seguirá un enfoque de filtro de partículas[3], muy utilizado en el mundo de la conducción autónoma por ser el método que mejores resultados está dando a día de hoy.

1.2 Objetivos

El objetivo del trabajo es iniciar una línea de investigación en el CSIC en cuanto a detección de ocupación del entorno. Se trata de informatizar la creación de mapas que contengan las posiciones de los objetos cercanos al vehículo y su velocidad. En cuanto a la implementación, no se debe perder la perspectiva de que el software se ejecutará en tiempo real[4], por lo que los algoritmos deben estar diseñados para ser tanto eficientes como paralelizables[5].

Como datos de entrada, cuento con los suministrados por el láser y el GPS de alta precisión. Para lograr el objetivo final -modelizar los obstáculos de la vía- se propusieron tres objetivos intermedios. El primero era lograr modelar la ocupación tomando como datos solo los suministrados en ese instante. Es decir, utilizar solo la información de lo que el coche "ve". En segundo lugar, se combina esta información instantánea con la recibida

anteriormente, para conseguir mejores estimaciones y ser capaz de modelar zonas inobservables en este instante, pero de las que sí he recibido cierta información en instantes anteriores. Para lograr esto, se utilizará la aproximación del filtro de partículas, muy popular por ser la aproximación más precisa y eficiente que existe hasta ahora. Así, el coche también "recuerda" las predicciones anteriores. Por último, se dotará de significado a la ocupación del mapa, agrupando zonas de alta ocupación y calculando sus velocidades, logrando así modelar obstáculos en la vía. El coche "infiere" qué hay en su camino y "predice" su posición en instantes posteriores.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Introducción**. Se describe brevemente la motivación para llevar a cabo este proyecto y los objetivos perseguidos. También se dan algunas nociones de notación.
- **Estado del arte.** En esta breve sección se da un pequeño repaso a otras técnicas similares a la utilizada en este trabajo y otros estudios existentes sobre el tema. Explico mi aportación al mundo de la detección de obstáculos.
- **Diseño.** La parte principal de la memoria. En esta sección se detallarán los distintos algoritmos utilizados para construir el software. Se divide en tres secciones principales. La primera es la detección instantánea, dónde se crea una rejilla de ocupación solamente a partir de los datos suministrados por el láser en cierto instante. La segunda parte es el filtro de partículas, dónde se combina la información de cierto instante con la de instantes anteriores. También se comienzan a dar algunas nociones (a nivel celda) de las velocidades. La última parte es el clustering[6], gracias al cual se podrán distinguir objetos y sus velocidades, así como mejorar el filtro de partículas inicial.
- **Desarrollo.** En esta breve sección, se expone cuál ha sido el *modus operandi* a la hora de desarrollar el software.
- **Integración, pruebas y resultados.** Aquí se detalla cuál ha sido el proceso de pruebas del software, y cómo han ayudado a debuguear[7] y mejorar la aplicación. También se incluyen una serie de resultados logrados en situaciones de conducción tanto controladas como reales.
- **Conclusiones y trabajos futuros.** En esta última parte se hará un pequeño repaso de todo lo desarrollado en este proyecto, además de proponer posibles mejoras y expansiones de este trabajo.
- **Anexos.** Incluyen información relevante acerca del proyecto.

2 Estado del arte

2.1 Contexto del trabajo

Hoy en día, uno de los aspectos más importantes de los Sistemas Avanzados de Asistencia al Conductor[9] (Advanced Driver Assistance Systems, ADAS) es la habilidad de reconocer los alrededores del vehículo. Distintos grupos de investigadores han propuesto sus distintos puntos de vista para reconocer peatones, identificar señales de tráfico o evitar colisiones, entre otros.

Sin embargo, el uso de ADAS está limitado hoy en día a maniobras a bajas velocidades y asistencia en el aparcamiento. Estas situaciones son relativamente sencillas ya que el sistema de percepción debe percibir pocos elementos, que están muy bien definidos y delimitados en entornos estructurados y principalmente estáticos. Se está investigando para extender su uso a situaciones más complejas en escenarios urbanos dinámicos con cierto tráfico.

Para lograrlo, los datos de los sensores (generalmente ruidosos) tienen que recoger uno o más potenciales obstáculos en el entorno del vehículo para realizar una estimación robusta en cada instante de tiempo de sus posiciones y velocidades. En otras palabras, es un problema de seguimiento de múltiples objetivos.

La aproximación clásica consiste en llevar a cabo un seguimiento de los obstáculos detectados en la vía, manteniendo una lista de los objetos identificados. Las principales dificultades residen en que se necesita conocimiento de (a) si una nueva observación de algún sensor corresponde a un objeto ya identificado o no, y (b) qué objetos deben introducirse, mantenerse o eliminarse de nuestra base de datos de potenciales obstáculos. Esto se vuelve impracticable en escenarios de mucho tráfico, dónde la aparición, desaparición y oclusión de elementos con velocidades variantes ocurren con demasiada frecuencia.

El uso de una rejilla 2D para la representación del espacio proporciona un marco de trabajo excelente para llevar a cabo la fusión de la información proporcionada por los distintos sensores [2]. Pueden especificarse distintos modelos de representación de la información de los sensores para adaptarlos a sus características específicas, facilitando la fusión de los datos en la rejilla. Prescindir de una representación basada en objetos facilita la fusión de información a bajo nivel, pudiéndose llevar a cabo de una manera más eficiente. Existen varias propuestas basadas en la teselación del entorno, como los documentos de Elfes [3] y Moravec [2], que definieron el modelado del input de los sensores para la rejilla.

En el artículo [4] se presenta el Filtro de Ocupación Bayesiana (Bayesian Occupancy Filter, BOF) que evalúa la ocupación del entorno sin realizar suposiciones sobre su

naturaleza. El objetivo principal del BOF consiste en permitir la combinación de información de múltiples sensores, para producir un sistema de reconocimiento del entorno más robusto. Algunos sensores que perciben el entorno (LIDAR, sensores estéreo, radares…) proporcionan miles de puntos en el espacio. Como ADAS requiere una respuesta en un lapso de tiempo muy pequeño, será necesario reducir las dimensiones de los datos recibidos sin comprometer el rendimiento del sistema. Este problema se aborda con una aproximación basada en rejilla con una formulación probabilística, propuesta inicialmente por Elfes en [3], que planta las semillas del método BOF. Este realiza un manejo intrínseco de la incertidumbre usando el Teorema de Bayes, permitiendo manejar ruido e información imprecisa de los datos suministrados por los sensores. Además, esta información se refina iterativamente utilizando información de instantes anteriores.

Durante estos últimos 15 años, varios grupos de investigadores han propuesto algoritmos que tratan de mejorar puntos débiles del BOF. Algunos ejemplos son CMCDOT (Conditional Monte Carlo Dense Occupancy Tracker) [5], propuesto por el centro de investigación INRIA, el filtro de partículas PHD/MIB [6], del equipo de Dominik Nuss, o el enfoque de BMW [7].

En la actualidad, hay mucha investigación llevándose a cabo en el ámbito de la detección de obstáculos. Sin embargo, ésta se camufla con secretismo debido a los intereses y las rivalidades de los investigadores y desarrolladores, especialmente cuando esta se utiliza para la conducción autónoma, un campo en pleno desarrollo y que en un futuro no muy lejano puede proporcionar muchos ingresos.

2.2 Aportación de este TFG

En este trabajo, se ha implementado un software capaz de detectar y modelar los distintos obstáculos que un coche autónomo (o cualquier entidad con el suficiente hardware) pueda encontrar en su camino. A partir de la nube de puntos captada por un láser y la pose proporcionada por un GPS de alta precisión, el software es capaz de inferir distintas informaciones de interés como la ocupación del entorno, los objetos que hay en él o la velocidad de los mismos. Para visualizar estos datos, este proporciona como salida una serie de imágenes y gráficas en donde quedan reflejados. Este trabajo inicia una línea de investigación en el CSIC para lograr lo que tantos grupos de investigación y empresas automovilísticas ansían: un software capaz de digitalizar la topología del entorno de la manera más precisa posible.

***Figura 2-1**: Coche autónomo del CSIC y algunos de sus dispositivos.*

El software se ha desarrollado inspirándose en una serie de papers citados a lo largo del documento. Aun así, en estos no se encuentran todos los detalles, y es cuando ha entrado en juego la imaginación y el buen criterio. Por otro lado, algunos módulos del software no están basados en ningún trabajo previo (como por ejemplo las transformaciones de rejillas o toda la algoritmia del clustering), sino que se han desarrollado originalmente, realizando labores de investigación puras. Este cóctel de información de distintas fuentes fruto del estudio de los trabajos de los expertos y de la meditación y desarrollo propios ha dado como resultado este trabajo. Se ruega al lector que le preste la atención que merece.

3 Diseño

3.1 Antes de empezar…

Esta sección será la parte principal de la memoria. Aquí se recoge el diseño de los algoritmos desarrollados, además de la cadena de sucesos ejecutados por el software para modelar el entorno.

En primer lugar, es importante enfatizar que las operaciones se realizan sobre una rejilla[9] de ocupación. Esto es, se está proyectando verticalmente el espacio en 2D, y luego se discretiza. Para ello, se divide el espacio en celdas pequeñas (20x20cm por ejemplo), y luego se calcula la probabilidad de ocupación de cada una de estas celdas. La probabilidad de ocupación de cada celda se considera independiente de la de las demás, pues de otra manera se necesitaría una capacidad de cómputo mucho mayor. Combinando finalmente las probabilidades de ocupación de cada una de estas celdas, se construye un mapa bidimensional de ocupación.

El software se divide en tres fases. La primera de ellas es la detección instantánea. Gracias a ella, se puede crear una rejilla de ocupación del instante presente, esto es, de un barrido del láser. La segunda fase es el filtro de partículas, que se utilizará para combinar la detección instantánea con otros mapas calculados en instantes anteriores, mejorando los resultados. Además, permitirá inducir velocidades a nivel de celdillas. En la última fase, se lleva a cabo clustering que permitirá modelar los objetos de la vía (coches y personas mayoritariamente) y sus velocidades. También permitirá mejorar el filtro de partículas en algunos aspectos que se discutirán más adelante. Estos tres pasos se ejecutarán una vez por cada barrido del láser. El filtro de partículas, a diferencia de las otras fases, se ayudará de los cálculos obtenidos en instantes anteriores.

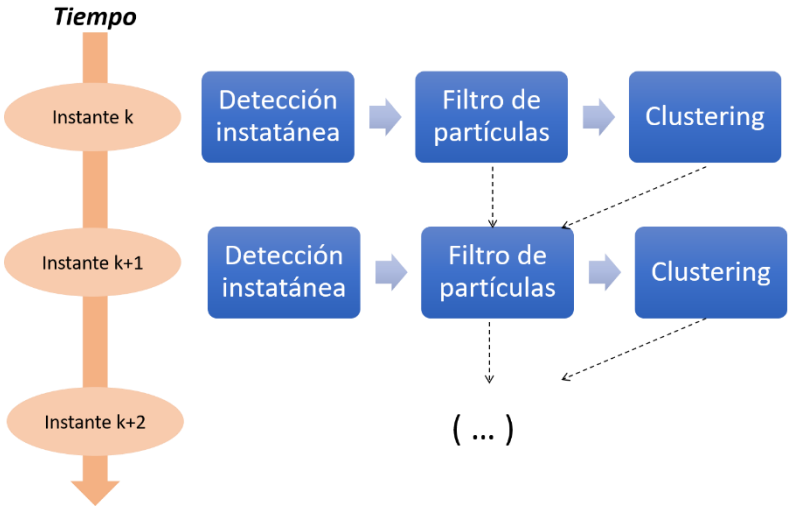

***Figura 3-1**: Flujograma del software*

3.2 Detección instantánea

3.2.1 Objetivo

El objetivo de la detección instantánea es procesar la nube de puntos[10] del láser en un barrido para construir una rejilla de ocupación del instante en cuestión.

***Figura 3-2**: Tecnología láser para detección de obstáculos en la conducción autónoma.*

Para crear esta rejilla, se utilizará la aproximación del Pool de opinión [8], usada para combinar información de distintos sensores. En este caso, aunque solo se cuenta con un sensor -el Lidar[11] este nos suministra 4 fuentes de información, que se corresponden con las cuatro capas con las que este cuenta. Información adicional del Lidar viene suministrada en el Anexo D. Esta aproximación inicial nos dará algunos problemas, debido a que un falso impacto[12] afectará enormemente al mapa final (ya que solo se cuenta con 4 sensores). Por lo tanto, se han desarrollado técnicas para mejorar los resultados del Pool. En la última sección de este apartado, se detallará como suministrar información no solo de la ocupación de la rejilla sino de la certeza o verosimilitud de la medida en cada celda.

3.2.2 Pool de opinión

El Pool de opinión consiste en la combinación de opiniones de varios sensores, utilizando también la confianza o verosimilitud de cada uno de ellos. En este caso, se busca calcular la probabilidad de ocupación de cada celda de nuestra rejilla. Así, esta probabilidad viene dada por la siguiente fórmula:

$$\mathrm{P}(\mathrm{C}|Y_1, Y_2, \dots, Y_m) = \frac{1}{\beta} \sum_{i=0}^{m} w_i(C) P(C|Y_i)$$

Donde β es un factor de normalización:

$$\beta = \sum_{i=0}^{m} w_i(C)$$

Esto es básicamente una media ponderada de las estimaciones de probabilidad de ocupación de la celda de cada sensor (en nuestro caso, las capas del Lidar, o sea que *m=4*). El término $P(C|Y_i)$ es la probabilidad de que la celda C esté ocupada medida por el sensor *i-ésimo*. $w_i(C)$ es la verosimilitud de esta medida, que actúa de ponderador, dando más peso a aquellas medidas con más verosimilitud. Por último, β es el factor normalizador, para que la probabilidad quede en el intervalo [0,1].

Por ejemplo, supongamos que una capa impacta completamente con una pared. Así, todas las medidas de ocupación de celdas de esa capa detrás de tal pared deberán tener verosimilitud muy baja o nula, ya que esa capa no puede obtener información fiable en esas celdas, y por tanto contribuirá poco al sumatorio total, siendo las medidas del resto de capas las que predominen.

Puede darse el caso de que todas las capas impacten completamente con una pared. Si sucede, todas las verosimilitudes tenderán a cero, y β podría llegar a valer 0, quedando la ecuación indeterminada. En este caso, se define que $\mathrm{P}(\mathrm{C}|Y_1, Y_2, \dots, Y_m) = 0$

Para llevar a cabo una implementación cómoda del pool de opinión, se define una rejilla polar. Esto es, cada celda cubre cierta porción del área a estudiar, indexadas por distancia y ángulo. La rejilla tendrá la siguiente forma:

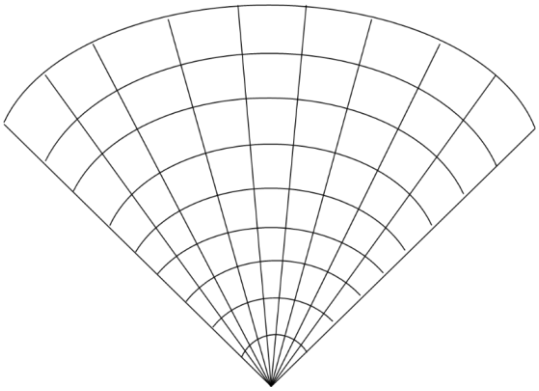

***Figura 3-3**: Rejilla polar*

Esto permitirá definir una serie de funciones que dependen de la distancia, pudiéndose aplicar a las columnas de la rejilla fácilmente, simplemente discretizando esta función. Más adelante se abordará el problema de transformar esta rejilla polar a una rejilla cartesiana.

Pero antes, se muestra como calcular la probabilidad de ocupación de cada celda y su verosimilitud.

3.2.2.1 Cálculo de la probabilidad de ocupación.

Se define una función de probabilidad de ocupación por cada capa y cada resolución angular[13]. En el caso de tener un solo impacto, se tendrá probabilidad de ocupación 0 hasta éste. Alrededor del mismo, se define una gaussiana (ya que el impacto puede tener cierto error), y más allá del impacto se fijará probabilidad 0.5, ya que no se tiene visibilidad en esta

zona y por tanto no se puede decidir nada en ella. La siguiente es la fórmula de probabilidad de ocupación respecto a la distancia:

$$P(x) = \begin{cases} f(x) & si\ x < x_0 \\ \max[\, f(x)\,,\ 0.5\,] & si\ x \geq x_0 \end{cases}$$

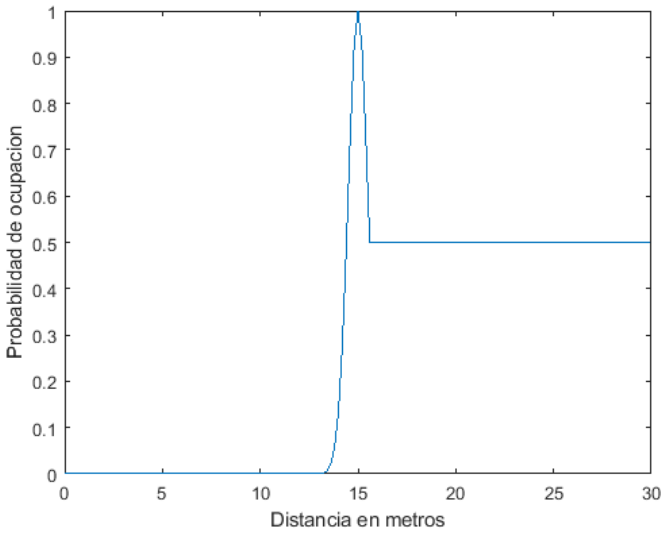

***Figura 3-4**: Probabilidad de ocupación de una dirección con impacto a los 10 metros.*

En la expresión anterior, $f(x)$ es la gaussiana de media la distancia de impacto (x_0) y de desviación típica σ, que es una decisión de diseño. Es decir:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-x_0)}{2\sigma^2}}$$

Si se tuvieran más impactos, como por ejemplo dos, se considera que a partir del primero se tiene $1/2$ de visibilidad, y que en la mitad observable no se observa masa (hasta el impacto), luego en este tramo se fija la probabilidad de ocupación a $1/4$. Si se tuvieran aún más impactos, se iría escalonando la probabilidad de ocupación entre impactos. Si se tuviese una sucesión de impactos $\{\, x_1, x_2, \dots\ ,\ x_n\}$, ordenados crecientemente, la expresión quedará:

$$P(x) = \begin{cases} f(x) & si\ x < x_1 \\ \max[f(x)\,, i/2n] & si\ x_i < x \leq x_{i+1} \quad i = 1, \dots, n-1 \\ \max[f(x)\,, n/2n] & si\ x_n < x \end{cases}$$

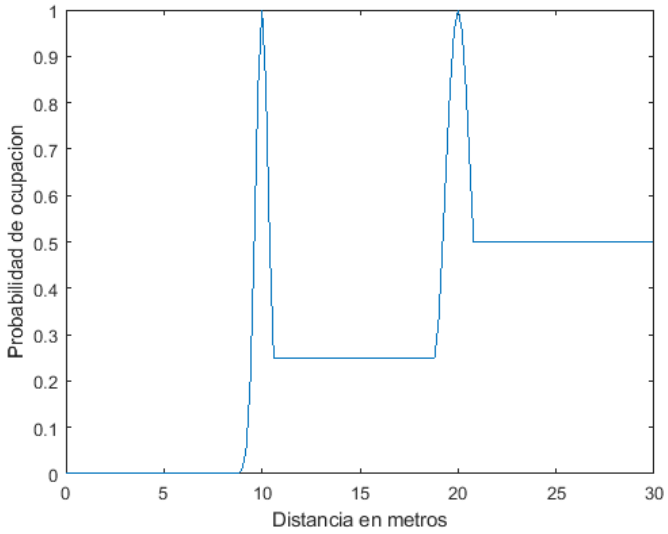

***Figura 3-5**: Probabilidad de ocupación con dos impactos a los 10 y 22 metros*

Discretizando estas funciones para cada celda en distancia se obtiene el cálculo de la probabilidad de ocupación de cada celda medida por una capa.

3.2.2.2 Cálculo de la verosimilitud de la medida

Para realizar este cálculo, se deben tener en cuenta dos factores que pueden afectar la verosimilitud de una medida del Lidar. En primer lugar, más allá de un impacto, la verosimilitud debe caer pues la visibilidad de la capa más allá del impacto es menor que antes o nula. Por otro lado, es deseable que las capas que impactan con el suelo tengan verosimilitud 0 al impactar con este. Si no, se tendrán impactos con verosimilitudes positivas en estas zonas y se deduce que esa celda está ocupada con un objeto relevante, cuando realmente lo que se detecta es el suelo. Al haber dos factores que afectan a la verosimilitud, se definen dos funciones independientes de verosimilitud, que luego se combinarán para dar el resultado final.

$$w_i(\mathrm{C}) = w_{i,inc}(\mathrm{C}) * w_{i,impacto}(\mathrm{C})$$

El primer factor será la verosimilitud de inclinación, cuyo valor bajará según la medida se tome más próxima al suelo. El segundo factor es la verosimilitud de impacto, que determinará como celdas inobservables (y por tanto, sin verosimilitud) a aquellas por detrás de un impacto. Se realizará el cálculo de cada tipo de verosimilitud por separado.

3.2.2.2.1 Cálculo de la verosimilitud de inclinación

Se define una función de verosimilitud de inclinación por capa. Como ya se ha mencionado, lo deseable sería que en las celdas correspondientes a impactos de la capa del láser con el suelo, la verosimilitud sea cero. Así, suponiendo conocida tal distancia (se

discute como conocerla en detalle en el Anexo F), se define una caída lineal de la verosimilitud hasta esta. Puede darse que una de las capas no impacte con el suelo totalmente sino solo parcialmente (por ejemplo, si la inclinación de la capa es de -0.4° y su anchura de 0.8°, solo el 50% de la capa impacta). En este caso, se tendrá una caída lineal cuyo valor en la distancia máxima será el porcentaje de la capa que no impacta con el suelo. Así, para cada capa i, se tiene la función:

$$w_{i,inc}(\mathrm{x}) = \begin{cases} \max\left[1 - \frac{x}{imp(i)},\ 0\right] & si\ i\ impacta\ completamente\ (a\ distancia\ imp(i)) \\ \alpha(i) + \left(1 - \alpha(i)\right)\left(1 - \frac{x}{d_{max}}\right) & si\ i\ impacta\ parcialmente\ (en\ una\ fracción\ \alpha(i)) \\ 1 & si\ i\ no\ impacta \end{cases}$$

A continuación, se presenta una posible gráfica para el Lidar de 4 capas, donde la capa roja impacta a los 10 metros, la verde a los 20, y la amarilla impacta parcialmente en un 22% (78% liberada). La capa azul, que es la superior, no impacta nunca.

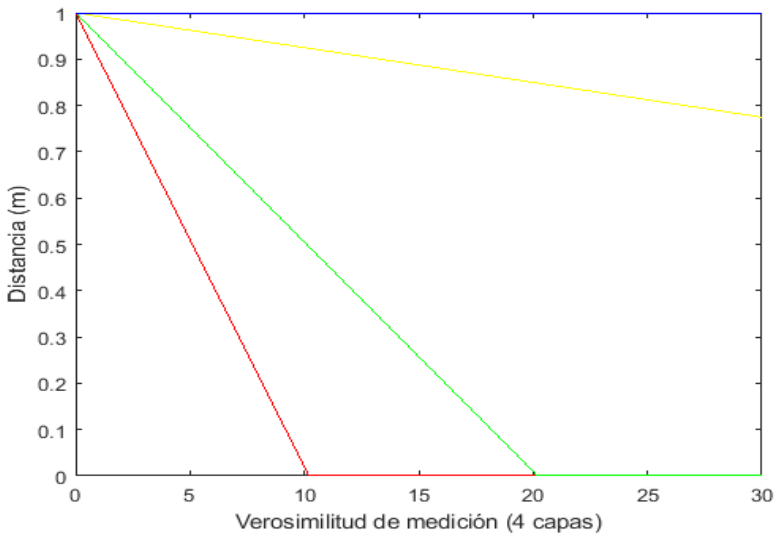

***Figura 3-6**: Verosimilitud de inclinación*

3.2.2.2.2 Cálculo de la verosimilitud de impacto

Se define una función de verosimilitud por capa y por resolución angular. La verosimilitud será 1 hasta el punto de impacto, a partir del cual la verosimilitud caerá a cero suavemente siguiendo una gaussiana, pues no se tiene visibilidad en esta zona:

$$w_{impacto}(x) = \begin{cases} 1 & si\ x < x_0 \\ f(x) & si\ x \geq x_0 \end{cases}$$

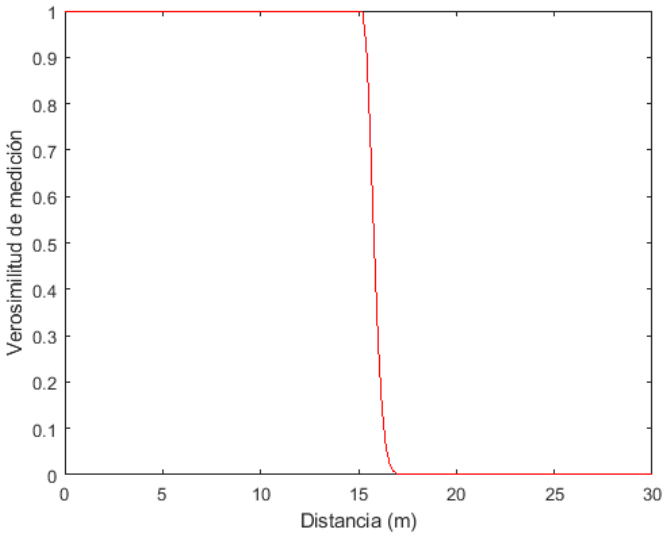

***Figura 3-7**: Función de verosimilitud de impacto, con impacto a los 15 metros*

Siendo $f(x)$ la gaussiana descrita en la sección 3.2.2.1, cuya desviación típica es un parámetro distinto del software.

Si se tienen varios impactos, siguiendo con la misma lógica de la Sección 3.2.2.1, se irán definiendo escalones que decaen en cada sector, modelando que la visibilidad se va reduciendo. Queda pues:

$$w_{impacto}(x) = \begin{cases} 1 & si\ x < x_1 \\ max\left[\frac{n-i+1}{n} f(x)\ ,\ \frac{n-i}{n}\right] & si\ x_i < x \leq x_{i+1} \quad i = 1, \dots, n-1 \\ \frac{1}{n} f(x) & si\ x_n < x \end{cases}$$

Muestro a continuación un ejemplo de la gráfica para dos impactos:

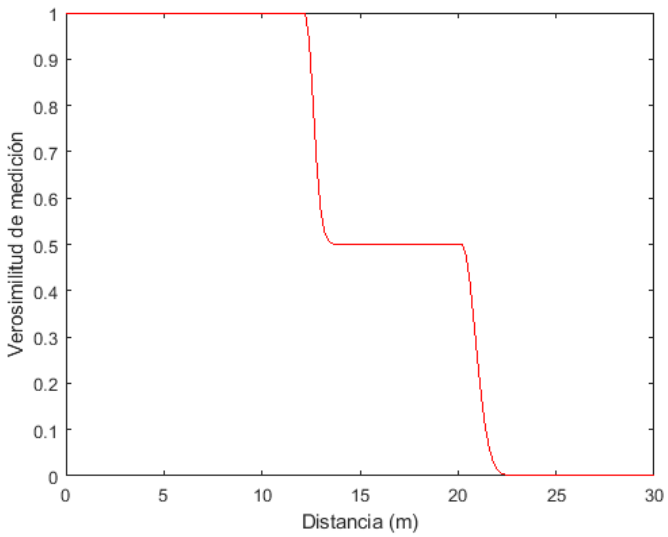

***Figura 3-8**: Función de verosimilitud de impacto, con impacto a los 10 y a los 20 metros*

3.2.2.3 Combinando los valores hallados en una rejilla cartesiana.

Una vez calculadas la probabilidad de ocupación medida por cada capa y su respectiva verosimilitud, ya se puede crear la rejilla polar de ocupación.

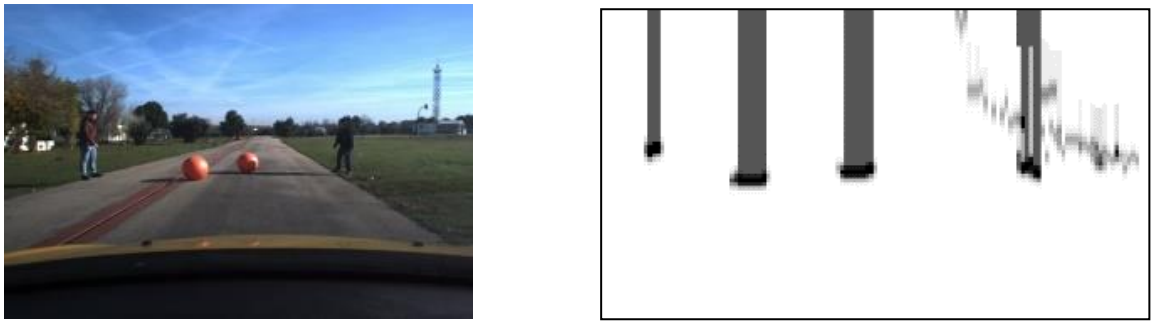

***Figura 3-9**: Instante capturado y mapa de ocupación polar instantáneo*

Sin embargo, una rejilla polar no es deseable pues el tamaño de cada celda no es uniforme, lo que podría ser problemático. Por ello, se busca transformar esta rejilla polar en una cartesiana. Esta transformación no es trivial, pues no hay una correspondencia uno a uno.

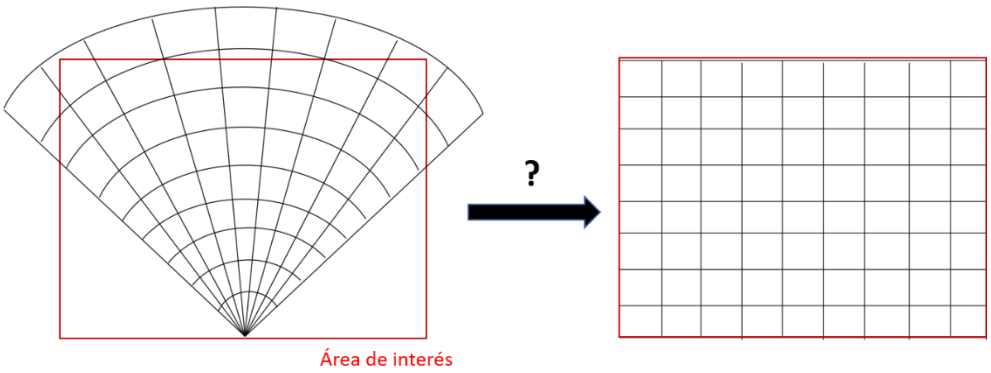

***Figura 3-10**: El problema de transformar de una rejilla polar a rejilla cartesiana*

Así, se ha ideado el siguiente algoritmo (Anexo C para su representación gráfica), que asociará a la rejilla cartesiana la media de los valores de las celdas polares que mejor lo aproximan. Para ello, se dividen las celdas de la rejilla cartesiana en 3 grupos:

1. **Celdas inalcanzables**: Son las celdas que no entran dentro del rango de visión de la rejilla polar, ya sea por las limitaciones angulares o por estar a una distancia superior a la máxima distancia de la rejilla polar. A estas celdas se les asocia una probabilidad de ocupación de 0.5 y una verosimilitud de la medida de 0.
2. **Celdas alcanzables "buenas"**: Son casi todas las celdas que entran en el campo de visión. Para rellenar estos valores, se calculan los puntos medios de las celdas polares. Luego, se observa a qué celdas de la rejilla cartesiana pertenecen. Así, el

valor de una celda cartesiana será la media de los valores de las celdas polares cuyos puntos centrales caen en ellas.

3. **Celdas alcanzables "malas"**: Son aquellas celdas alcanzables que no tienen ningún punto central de ninguna celda polar en su área. Para calcular sus valores, se hace una media de los valores de las celdas adyacentes alcanzables buenas, suavizando así los valores en los huecos que deja el algoritmo.

Una duda que surge inmediatamente es si habrá muchas celdas "malas", lo que haría que el algoritmo no fuese tan efectivo. La respuesta es que a distancias cortas (siempre dependiendo de las resoluciones de cada rejilla) no, pero a medida que uno se aleja del origen, aumentan estas celdas indeseables. En el Anexo C se discutirá un poco más este tema. De momento basta decir que con las resoluciones que se ha trabajado, no se tiene problema alguno con la densidad de celdas de este tipo hasta unos 40 metros aproximadamente.

Usando el método antes descrito, se consigue transformar la rejilla polar de ocupación en una rejilla cartesiana de ocupación, logrando nuestro objetivo principal.

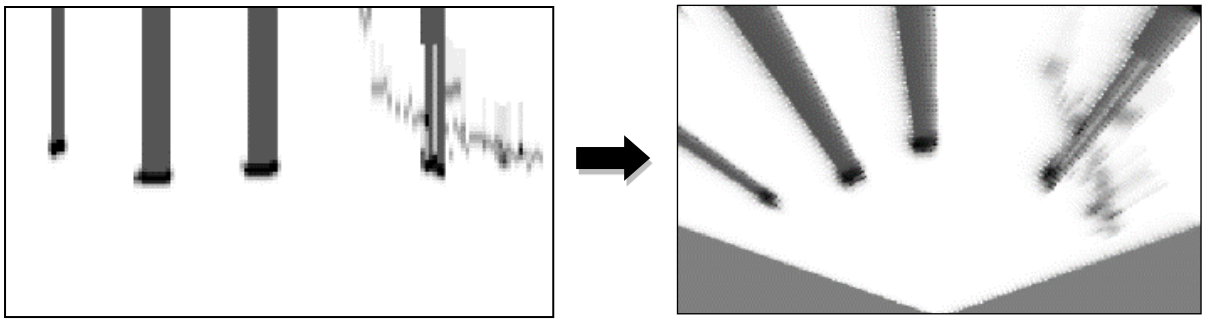

Figura 3-41: *Ejemplo de transformación de rejilla polar a cartesiana*

Sin embargo, esta aproximación no es perfecta. En nuestro caso, solo se cuenta con cuatro capas de información, con lo que un falso positivo generará más ruido del deseable. Para evitarlo, se han desarrollado técnicas novedosas que tratan este problema. Por un lado, se calcula la distancia a la que cada capa impacta con el suelo en cada iteración, logrando distinguir qué impactos se deben a la carretera, y por lo tanto no corresponden a obstáculos reales (Anexo E). Así, la distancia de impacto mencionada en la Sección 3.2.2.2.1 se calcula dinámicamente. Por otro lado, se filtra la nube de puntos, despreciando aquellos impactos que debido a su posición inesperada, son candidatos a ser falsos positivos (Anexo F). En el Anexo G, se muestra el resultado de aplicar estas correcciones a la detección instantánea.

3.2.3 Aportando información sobre certidumbre.

Hasta ahora se ha construido un mapa de probabilidad de ocupación (instantáneo). Sin embargo, es importante también aportar información sobre la certidumbre de tal ocupación. No es lo mismo una celda con probabilidad de ocupación 0.5 porque no ha sido observada, que otra con probabilidad de ocupación 0.5 debido a que dos capas impactaron con un objeto y dos capas no lo hicieron. Para modelar esta certidumbre, se utiliza el concepto de masa de evidencia. Para los desconocedores de la teoría de masas de evidencia, en el Anexo I se definen y explican brevemente algunos conceptos básicos.

Se define una masa de evidencia, denominada masa observada M_{obs}. Para que quede totalmente definida, bastará definir $M_{obs}(O)$ y $M_{obs}(L)$, denotando O para ocupada y L para libre. Para calcular estos valores, se define la certidumbre ($\hat{C}$) de una medida en una celda de la siguiente manera:

- Si $\sum_{i=0}^{m} w_i(C) \geq umbral$, se decide que la certidumbre en esa celda es absoluta, es decir, $\hat{C}(C) = 1$. Esto equivale a decir que en las celdas donde se tenga al menos *umbral* capas operativas, tendré certidumbre total de las medidas que me otorguen. (En la práctica, se utilizará *umbral* = 2, que es la mitad de las capas, porque es una cantidad razonable de verosimilitud, y diversas pruebas han mostrado que funciona. Aun así, es un parámetro del software).
- Si $\sum_{i=0}^{m} w_i(C) < umbral$, habrá una caída lineal de la certidumbre hasta llegar a 0 cuando la suma de las verosimilitudes sea cero. Así, $\hat{C}(C) = \frac{1}{umbral} \sum_{i=0}^{m} w_i(C)$

Una vez calculada la certidumbre de una celda, se calcula la masa observada:

$$M_{obs}(O) = P(C) * \hat{C}(C)$$

$$M_{obs}(L) = (1 - P(C)) * \hat{C}(C)$$

Así, áreas inobservadas tendrán ambas masas igual a cero por tener certidumbre cero, mientras que celdas donde dos capas impactan y dos no, tendrán certidumbre uno y masas igual a 0.5. A continuación se muestra un ejemplo. De izquierda a derecha: instante capturado, probabilidad de ocupación, masa ocupada y masa libre, correspondiendo a masa 0 el color blanco y masa 1 el color negro. Observar que donde la certidumbre es 0 (detrás de las personas y las pelotas), tanto la masa ocupada como la masa libre valen 0, al igual que en las regiones inferiores no observadas por el Lidar. Sin embargo, la probabilidad de ocupación en estas regiones es 0.5 (tonalidad gris).

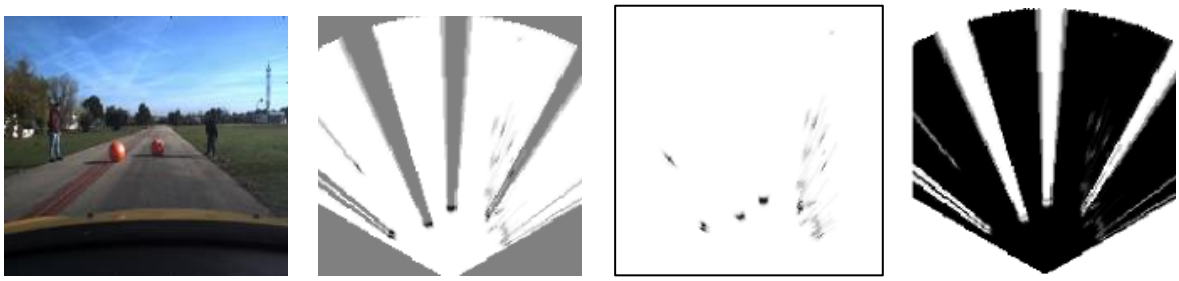

***Figura 3-12**: Ejemplo de rejilla mostrando probabilidad y masas de ocupación.*

3.3 Filtro de partículas

3.3.1 Objetivo

Hasta ahora, se ha conseguido modelar la ocupación del entorno a partir únicamente de las mediciones del Lidar en ese instante. Sin embargo, se puede tener en cuenta más información, por ejemplo, los mapas de ocupación de instantes anteriores. Esta información se puede utilizar para mejorar la estimación de la ocupación, para estimar ocupación en áreas inobservables (por oclusión o por rango de visión) pero que fueron observadas en instantes anteriores, e incluso para estimar las velocidades de las celdas. Para lograr todos estos objetivos, se utiliza una aproximación bastante usada en el mundo de la conducción autónoma: un filtro de partículas.

3.3.2 Idea general

Un filtro de partículas, como su propio nombre indica, utilizará partículas para representar la masa ocupada (se recuerda que la masa ocupada observada se recalcula con cada barrido del Lidar). Estas partículas tienen posición y velocidad (2 coordenadas para cada una pues el mapeo es en 2D) y un peso. A más peso, más masa representa la partícula. No debe confundirse esta masa con el sentido de masa en el ámbito de la física. Esta masa representa la certidumbre de que existe algún obstáculo allá dónde se encuentre la partícula. Estas partículas se moverán por la rejilla, tratando de predecir la distribución de la masa en los próximos instantes.

Así, se implementará un algoritmo iterativo, donde cada iteración corresponderá a un barrido del Lidar. Para cada iteración, se calculará la masa observada M_{obs} y la masa predicha M_{pred}, inferida a partir del movimiento de las partículas. Se utilizará luego la combinación de masas de evidencia de Dempster-Shafer [9] para calcular la masa actualizada M_{act}, que será la masa de evidencia combinada y final para esa iteración. Además, las velocidades de las partículas darán una idea de las velocidades de las celdas y, por tanto, de la masa de nuestro entorno.

El número de partículas será clave en el funcionamiento del algoritmo. A más partículas, mayor precisión de estimación de la ocupación y de las velocidades de las celdillas, pero más tiempo de procesado necesitará el algoritmo. Como el software tiene como objetivo funcionar a tiempo real, no se puede permitir que el tiempo que tarda en procesar la información varíe demasiado. Por ello mismo, el número de partículas está fijado.

3.3.3 Algoritmo en siete pasos

Todo algoritmo basado en el filtro de partículas se divide en los mismos siete pasos:

1. **Definición de rejilla y predicción de partículas.** Se define el nuevo eje de coordenadas y la nueva rejilla compensando el momento del coche. También, se predice la nueva posición de las partículas.

2. **Asignación de partículas a celdas (cálculo de masa predicha).** Para cada celda de la rejilla, se recogen las partículas que caen en ellas. Así se puede calcular la masa predicha de cada celda.
3. **Combinación de masa observada y predicha.** Se combinan ambas masas usando las reglas de Dempster-Shafer para obtener la masa actualizada.
4. **Cálculo de los momentos de las celdas.** Se calcula la velocidad de las celdas de la rejilla.
5. **Actualización de los pesos de las partículas persistentes.** Se actualizan los pesos de las partículas persistentes en función de si se encuentran en celdas con más o menos masa de ocupación.
6. **Inicialización de partículas nuevas.** Se incializan partículas nuevas en lugares estratégicos para modelar la masa que aparece en lugares inesperados (entra en el campo de visión o surge en una zona imprevista).
7. **Remuestreo.** Se remuestrea las partículas para evitar degeneración del algoritmo, y conseguir que el número de partículas siga fijo a la próxima iteración.

A continuación, se explican más en detalle cada uno de los pasos arriba mencionados.

3.3.3.1 Definición de rejilla y predicción de partículas.

Lo primero que se debe hacer es compensar el movimiento del coche, definiendo un nuevo eje de coordenadas. Esto será posible pues el coche cuenta con un GPS de alta precisión, dotando posición y orientación. Por lo tanto, será sencillo calcular la diferencia de posición y de ángulo del coche respecto a la iteración anterior.

Primero se ha de calcular la nueva posición y velocidad de las partículas respecto al nuevo eje de coordenadas. Al tratarse de puntos en el espacio con cierto momento, bastará aplicarles una traslación y un giro para que queden representadas en el nuevo eje de coordenadas. Las partículas cuya posición resulta fuera del campo de visión actual, se olvidan, pues no aportarán información relevante y darán lugar a un procesado extra indeseable.

Habrá más dificultades al transformar la rejilla. Esto será necesario porque parte de la entrada de una iteración son los valores de la masa actualizada de la iteración anterior. A la hora de calcular estos valores, se tiene el mismo problema que al transformar la rejilla cartesiana a polar: no hay correspondencia clara uno a uno. Se seguirá la misma filosofía ya usada. Se calculan los puntos medios de la nueva rejilla, se calcula en qué celda se encuentran de la rejilla anterior, y se rellenan los valores de la nueva celda con la antigua correspondiente. Las celdas cuyo punto medio no se encuentra en ninguna celda antigua, son celdas inexploradas, y se les fija masa ocupada y libre cero.

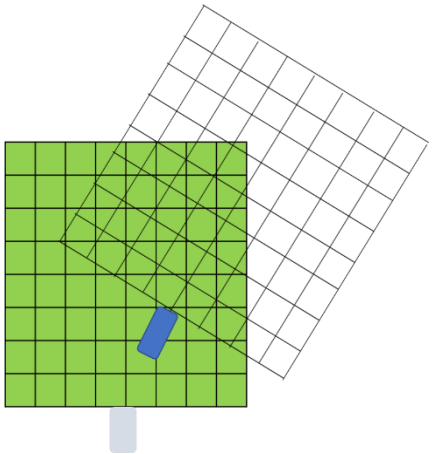

***Figura 3-53**: De nuevo el problema de transformar la rejilla*

Por último, se debe predecir la nueva posición de las partículas, aplicándoles su movimiento. Las que tienen una velocidad por debajo de cierto umbral (fijado a ~0.3m/s) se consideran estáticas,

y por tanto se fija su velocidad a cero y no se moverán. Para el resto, se supondrá que estas tienen movimiento uniforme. Bastará pues aplicarles este movimiento, añadiendo cierto ruido pequeño con distribución normal a la velocidad, $\varepsilon \sim \boldsymbol{N}^2\ (0,\Theta)$, para que así las partículas duplicadas (remuestreadas más de una vez) no se desplacen exactamente al mismo sitio, lo que haría inútil el hecho de tener dos partículas en vez de una. Además, esto ayudará al software a anticiparse a posibles cambios de ritmo, debido al gran número de partículas y a la pequeña componente aleatoria en sus velocidades. Por último, el peso predicho de cada partícula será el peso que arrastraba de la iteración anterior multiplicado por la probabilidad de supervivencia de una partícula (que es un parámetro del algoritmo, y está fijado a 0.99).

$$x_{k+1} = \begin{pmatrix} 1 & 0 & T+\varepsilon_x & 0 \\ 0 & 1 & 0 & T+\varepsilon_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} x_k \qquad w_{pred,k+1} = p_S * w_k$$

La expresión de la izquierda corresponde a la transición de estados (un estado está compuesto por cuatro números reales: posición en *x*, posición en *y*, velocidad en *x*, velocidad en *y*) de una partícula entre dos tiempos: *k* y *k+1*. El peso predicho viene definido por la fórmula de la derecha, siendo p_S la probabilidad de supervivencia de una partícula.

3.3.3.2 Asignación de partículas a celdas (cálculo de masa predicha)

Una vez hallada la nueva posición de las partículas, se calcula la masa predicha ocupada de cada celda. Para ello, simplemente se suman los pesos de las partículas que caen en ella. Si esta suma es superior a uno, se fija la masa predicha a uno, y se normalizan los pesos de las partículas para que su suma sea exactamente uno, dividiendo por la suma de los pesos de las partículas en esa celda.

$$\boldsymbol{M}^{\boldsymbol{C}}_{\boldsymbol{pred,\,k+1}}(\boldsymbol{O}) = \sum_{p\ en\ C} w_{p,\,k+1}$$

Por otro lado, para calcular la masa predicha libre, se sigue la suposición de que un espacio libre sigue libre, exceptuando por la posibilidad de que un objeto irrumpa en esa zona. Esto se modela con una función de difusión de masa predicha libre que dependerá del tiempo (cuánto más tiempo pase, más probabilidad de que tal objeto irrumpa, y por tanto más decrece la masa predicha). Por otro lado, debe estar acotada por uno menos la masa predicha ocupada, pues la suma de ambos no pueda ser superior a uno.

En resumen, la masa predicha se calcula siguiendo la siguiente expresión (siendo α(T) el factor de difusión de masa libre respecto del tiempo):

$$\boldsymbol{M}_{\boldsymbol{pred,\,k+1}}(F) = \min\left[\ \alpha(\mathrm{T}) * \boldsymbol{M}_{\boldsymbol{act},k(F)},\ 1 - \boldsymbol{M}_{\boldsymbol{pred},\,k+1}(O)\right]$$

3.3.3.3 Combinación de masa observada y predicha.

Llegados a este punto, ya se tiene tanto la masa predicha (calculada en el paso anterior) como la masa observada (proporcionada por la detección instantánea de la Sección 3.2) de la rejilla de esta iteración. Se calcula la masa actualizada de la rejilla usando las reglas de combinación de Dempster-Shafer (Anexo I).

$$M_{act} = M_{pred} \oplus M_{obs}$$

Esta masa actualizada será la masa final de esta iteración del algoritmo. Con ella, se puede dibujar completamente el mapa de probabilidad de ocupación y el mapa de masa ocupada (que realmente nos será más útil, pues realmente lo que se quiere modelar es la masa que se sabe que se encuentra en el entorno), consiguiendo parte de los objetivos del filtro de partículas.

Por otro lado, se dividirá la masa actualizada ocupada de cada celda en dos tipos de masa: la masa persistente $\boldsymbol{\varrho}_p$, que representa la masa que persiste desde el instante anterior, y la masa recién nacida $\boldsymbol{\varrho}_b$, que representa la masa que nace en este paso, ya sea para modelar objetos nuevos u objetos que surgen en lugares inesperados, quizás debido a un cambio de velocidad. La masa persistente total será la suma de las masas persistentes de cada una de las celdas, e igualmente con la masa recién nacida total.

Suponiendo que existe una probabilidad de nacimiento p_b, que es la probabilidad de que una partícula nazca en una celda libre (parámetro del algoritmo, será en torno a 0.02), se deduce la relación entre la masa persistente y la masa recién nacida.

$$\frac{\boldsymbol{\varrho}_{b\,k+1}}{\boldsymbol{\varrho}_{p,\,k+1}} = \frac{p_b * [1 - M_{\boldsymbol{pred},\boldsymbol{k+1}}(O)]}{M_{\boldsymbol{pred},\boldsymbol{k+1}}(O)}$$

Esto, combinado con la fórmula de descomposición de masa ocupada en persistente y recién nacida:

$$M_{\boldsymbol{act},\,\boldsymbol{k+1}}(O) = \boldsymbol{\varrho}_{p,\,k+1} + \boldsymbol{\varrho}_{b\,k+1}$$

Aporta las ecuaciones para calcular tanto la masa persistente como la recién nacida de cada celda (despejando):

$$\boldsymbol{\varrho}_{b,\,k+1} = \frac{M_{\boldsymbol{act},k+1}(O) * p_B * [1 - M_{pred,\,k+1}(O)]}{M_{pred,\,k+1}(O) + p_B * [1 - M_{pred,\,k+1}(O)]}$$

$$\boldsymbol{\varrho}_{p,\,k+1} = M_{act,\,k+1}(O) - \boldsymbol{\varrho}_{b,k+1}$$

3.3.3.4 Cálculo de los momentos de las celdas.

En este apartado, se muestra como calcular la velocidad de las celdas. Se realizará una media ponderada de las velocidades de las partículas en cada celda, ponderando por el peso de las propias partículas.

$$\boldsymbol{Vel}_{(x,y)}\,(\mathrm{C}) = \frac{1}{\beta} \sum_{part\ en\ C} (w_{pred,k+1}^{part} * v_{(x,y)}^{part})$$

Donde β es el factor de normalización:

$$\beta = \sum_{part\ en\ C} w_{pred,k+1}^{part}$$

En este paso se ha cumplido otro de los objetivos de esta sección: calcular la velocidad de las celdas. Los pasos restantes serán preparatorios para la siguiente iteración.

3.3.3.5 Actualización de los pesos de las partículas persistentes.

Las partículas persistentes son aquellas provenientes de la anterior iteración. Es decir, de momento, son todas las partículas que hay. Se debe calcular los pesos finales de estas partículas. Se normaliza los pesos de estas partículas en cada celda para que su suma sea la masa persistente en cada una de ellas. Por lo tanto, el peso de una partícula *p* persistente que se encuentra en una celda *C* será:

$$w_{k+1}^{p} = \frac{w_{pred,k+1}^{p} * \boldsymbol{\varrho}_{b,k+1}^{C}}{\sum_{part\ en\ C}^{-} w_{pred,k+1}^{part}} = \frac{w_{pred,k+1}^{p} * \boldsymbol{\varrho}_{b,k+1}^{C}}{M_{pred,k+1}^{C}(O)}$$

3.3.3.6 Inicialización de partículas nuevas.

En este paso, se inicializarán las partículas nuevas o recién nacidas, para representar la masa recién nacida en las celdas. Antes de hacerlo, se deben introducir una serie de parámetros.

En primer lugar, se debe definir la probabilidad de asociación p_A de una partícula a su celda, que es la probabilidad de que una partícula que nace en la rejilla esté asociada a un objeto ya existente, o corresponda a un objeto completamente nuevo, no detectado anteriormente. Ha de recalcarse que este parámetro es muy importante para el correcto funcionamiento del algoritmo. Por suerte, en posteriores secciones (3.4), se tendrá algún criterio para establecer este valor.

Esta probabilidad de asociación p_A divide las partículas que nacen en dos conjuntos: las partículas asociadas y las no asociadas. Cada una de ellas tendrá distintas distribuciones de probabilidad al nacer.

- Un nacimiento asociado vendrá dado por la distribución de probabilidad ***p(·)***
  - La distribución de posición es uniforme en la celda.
  - La distribución de velocidad será (de momento) la misma que la de la celda, con cierto ruido con distribución normal centrada en 0 y de poca variación. En la Sección 3.4.5.3 se sugiere una mejora.
- Un nacimiento no asociado tendrá una distribución de probabilidad ***b(·)***
  - La distribución de posición es uniforme en la celda.
  - La distribución de velocidad es:
    - Velocidad = 0 con probabilidad *p_quieta* (parámetro del algoritmo, ~0.1).
    - Velocidad uniforme (acotada a la velocidad máxima de la vía) con probabilidad *1 – p_quieta*.

.

Por último, se establece el número de partículas que nacen a cada iteración, denotado v_b. Un valor del 10% del total de partículas funciona bien (decisión de diseño).

Una vez fijados estos conceptos, se puede proceder. Se inicializan v_b partículas de manera aleatoria en las celdas, de manera proporcional a la masa recién nacida en cada una de ellas. Una vez calculadas cuantas partículas nacerán en cada celda, se dividen en partículas asociadas (A) y sin asociar (¬A).

$$v_{A,k+1} = round(p_{A,k+1} * v_{b,\,k+1})$$

$$v_{\neg A,\,k+1} = v_{b,\,k+1} - v_{A,\,k+1}$$

Cada partícula se inicializa con la distribución de probabilidad correspondiente, estableciendo sus posiciones y velocidades. Falta establecer su peso, que será para cada tipo de partículas (asociadas y sin asociar) el necesario para que la suma de todas ellas sea la masa recién nacida en la celda.

$$w_{A,\,k+1} = \frac{p_{A,\,k+1} * \boldsymbol{\varrho}_{b,\,k+1}}{v_{A,\,k+1}}$$

$$w_{\neg A,\,k+1} = \frac{[1 - p_{A,\,k+1}] * \boldsymbol{\varrho}_{b,\,k+1}}{v_{\neg A,\,k+1}}$$

Consiguiendo así inicializar el conjunto de partículas recién nacidas.

3.3.3.7 Remuestreo

Por último, se remuestrean las partículas para evitar degeneración. Así, se consiguen eliminar partículas que nos ofrecen poca información, y quedarse con aquellas más relevantes. Además, se controla el número de partículas que sobreviven a la iteración para que este sea fijo (v).

Se remuestrea por separado las partículas estáticas y dinámicas. La razón es que no tiene sentido tener más de una partícula estática en una celda, pues a la siguiente iteración

se tendría que procesar dos veces una partícula sin velocidad en el mismo sitio. Será mejor tener una solo partícula cuyo peso sea la suma de ambas, y así se ahorra una vuelta del bucle. Por lo tanto, el primer paso será separar las partículas en estáticas y dinámicas.

Las primeras partículas que se remuestrean son las estáticas. En primer lugar, se aglomeran las partículas estáticas que pertenecen a la misma celda. Esto es, se incializa una sola partícula en el centro de cada celda con alguna partícula estática, con velocidad cero, y con peso la suma de los pesos de las partículas estáticas en esa celda. Luego, se eliminan aquellas partículas con un peso ínfimo (por debajo de cierto umbral, fijado a 0.01) y restarán c partículas estáticas.

Se elige aleatoriamente y proporcionalmente a sus pesos una partícula dinámica, la cual sobrevive a esta iteración. Se repite esta operación $v - c$ veces, con reemplazamiento. De esta manera, se remuestrean en total v partículas.

Lo ingenioso de remuestrar por separado las partículas estáticas y dinámicas es que no habrá más partículas estáticas de las estrictamente necesarias, maximizando así el número de partículas dinámicas supervivientes, logrando una mejor estimación de las velocidades (debido a que se concentran más partículas en el movimiento) a la siguiente iteración.

3.3.3.8 Paso extra: inicialización del algoritmo.

El algoritmo supone que, para cada iteración, hay partículas y masa actualizada que provienen de la iteración anterior. Sin embargo, en algún momento se debe inicializar el algoritmo. Aunque cómo hacerlo no es fundamental para su correcto, es conveniente hacerlo lo mejor posible pues hará que el algoritmo converja a buenos resultados más rápidamente.

Lo que se hace es, en el primer barrido, inicializar las partículas y la masa actualizada a partir solamente de la masa observada. De hecho, la masa actualizada será directamente la masa observada, pues no se tiene evidencia de iteraciones anteriores para predecir nada. Por otro lado, se inicializarán v partículas con la distribución no asociada $\boldsymbol{b(\cdot)}$, distribuidas en las celdas proporcionalmente a la masa observada de cada una, y con peso $masa\_observada\_total/v$, de modo que la suma de los pesos sea la suma de las masas observadas en las celdas.

3.3.4 Resultados

Una vez implementado el algoritmo, se prueba su funcionamiento frente a distintos datasets[12], que se detallarán más en profundidad en la sección 5. El filtro de partículas mejora considerablemente la detección instantánea. Por un lado, permite recordar información de áreas oclusas pero que en cierto momento han sido visibles (debido al movimiento de los objetos o del propio coche). El mapa de probabilidad de ocupación, por tanto, mejora considerablemente. Por otro lado, este nos da una idea de las velocidades de las regiones con ocupación, determinada por la velocidad de las partículas en esta zona. En el Anexo J adjunto una serie de imágenes que representan gráficamente estas mejoras.

3.3.5 Posibles mejoras

Ya se han discutido los logros del filtro de partículas. Sin embargo, se puede llevarlo un paso más allá, ya que hay una serie de aspectos mejorables.

En primer lugar, se ha conseguido modelar la ocupación del entorno, pero no de los objetos como tal, que es algo mucho más útil para el coche autónomo. Sería deseable que las velocidades no estuvieran expresadas a nivel celdilla, ya que como se ha visto en el Anexo J puede ser bastante caótico y con mucha discrepancia entre celdas próximas. Si se pudiesen extraer objetos del mapa de ocupación, se podría calcular sus velocidades a partir de las de las celdas, y así de paso ver como de bien se están estimando.

Por otro lado, aparece bastante ruido en algunas celdas del mapa. Como el movimiento de las partículas tiene una componente aleatoria bastante considerable, esto puede llevar a la aparición de partículas en lugares inesperados, pudiendo producir ruido.

Por último, no hay ningún criterio para calcular las probabilidades de asociación p_A de las celdas. Tenerlo mejoraría considerablemente la eficiencia y precisión del algoritmo, ya que de momento se fija a un valor estático, que no se adapta a cada situación (celda) concreta.

Para solucionar todos estos problemas, se ha desarrollado la fase final de mi proyecto.

3.4 Clustering

3.4.1 ¿Qué es?

El clustering es una técnica que consiste en el agrupamiento de puntos (en clusters) por cercanía u otros criterios. En el caso de la conducción autónoma, se suele utilizar para pasar de una nube de puntos a una colección de objetos o clusters. En este caso concreto, no se busca agrupar puntos, sino celdas, usando a nuestro favor la masa de ocupación. En el ejemplo a continuación, mostramos la masa ocupada en cierto instante y el resultado de clusterizar este instante.

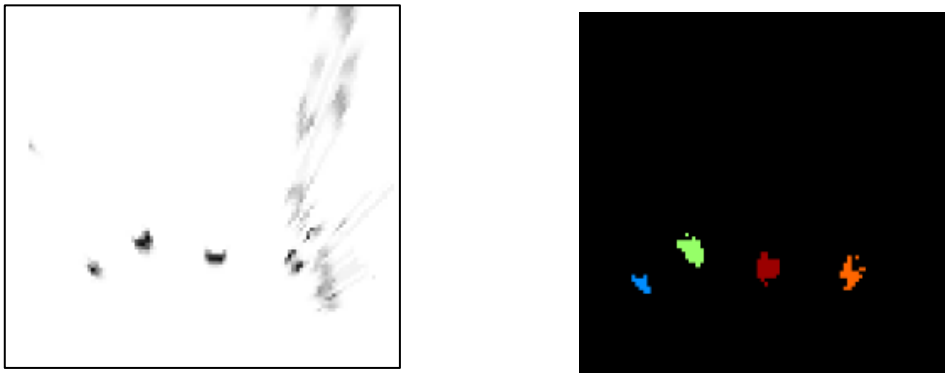

***Figura 3-64**: Ejemplo de rejilla clusterizada.*

3.4.2 ¿Qué ofrece?

El clustering aportará una serie de beneficios:

- En primer lugar, permitirá distinguir objetos en la vía, no sólo ocupación. Se podrán también inferir las velocidades de estos, y realizarles un seguimiento a lo largo del tiempo.
- Se mejorarám velocidades de las celdas mediante un reajuste inferido del movimiento de estos clusters. Esto permitirá usar menos partículas, lo que también hará que el algoritmo se ejecute más rápidamente.
- Permitirá eliminar ruido proveniente de celdas aisladas ocupadas.
- Aportará un criterio para calcular la probabilidad de asociación de cada celda.

Por otro lado, la única desventaja que podría ofrecer es un incremento del tiempo de ejecución. Sin embargo, esto no ocurre, sino que disminuye el tiempo de ejecución ya que el número de partículas para obtener resultados similares decrece cuantitativamente.

3.4.3 Algoritmo de clustering

3.4.3.1 DBSCAN

El algoritmo de clustering que se usa está basado en un algoritmo clásico: el DBSCAN (Density-based spatial clustering of applications with noise). Se basa en la densidad de los puntos (no las celdas) para agrupar estos mismos en clusters. Este requiere dos argumentos: el mínimo de puntos *minPoints* y la distancia de acción *ε*. El algoritmo clasificará los puntos en 3 grupos:

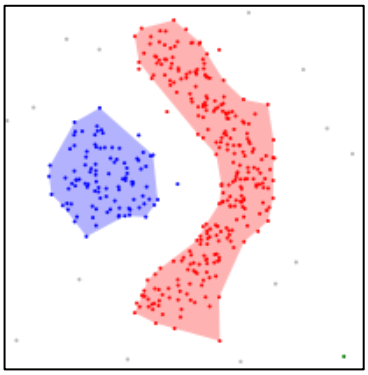

***Figura 3-75**: Clustering usando DBSCAN*

- Un punto p es un punto núcleo si al menos *minPts* puntos están a una distancia ε de él y, esos puntos son directamente alcanzables desde p. No es posible tener puntos directamente alcanzables desde un punto que no sea un núcleo.
- Un punto q es alcanzable desde p si existe una secuencia de puntos $p_1, p_2, \ldots p_n$ puntos donde $p_1 = p$ y $p_n = q$ tal que cada punto p_i es directamente alcanzable desde p_{i-1}; es decir, todos los puntos de la secuencia deben ser puntos núcleos, con la posible excepción de q.
- Un punto que no sea alcanzable desde cualquier otro punto es considerado ruido.

Un punto p, junto con todos los puntos directamente alcanzables desde él, formarán un cluster. En el Anexo K se incluye el pseudocódigo de este algoritmo.

Este algoritmo es considerado de los más robustos de su tipo. En primer lugar, DBSCAN no necesita de la especificación del número de clusters deseado como lo requieren otros algoritmos. Puede también encontrar clusters con formas geométricas arbitrarias, incluso hallar un cluster completamente rodeado (pero no conectado) de otro cluster distinto.

Tiene nociones de ruido, requiere solo de dos parámetros y no es susceptible al orden en que se encuentren los puntos dentro de la base de datos. Algunas de sus desventajas son que depende fuertemente de sus parámetros y de la distancia elegida, aunque esto no dará problemas si se hace una buena elección de los mismos mediante el estudio previo del dataset en cuestión.

3.4.3.2 DBSCAN suavizado.

Se busca ahora aplicar DBSCAN a nuestra rejilla. Una opción lógica sería definir un umbral de masa de ocupación a partir del cual se considera que la celda está ocupada. Una vez hecho esto, se tratan estas celdas ocupadas como puntos a clusterizar, sobre los que aplicar el DBSCAN clásico. En primera instancia, se implementó esta versión y funcionó bastante bien. Sin embargo, no se usa toda la información disponible: discretizar una variable como la masa de ocupación (un número real entre 0 y 1) como libre u ocupado es un mal paradigma pues se pierde bastante información. Sería deseable un algoritmo que permita utilizar esta información tal cual.

Y así nace el DBSCAN suavizado. Este utilizará también dos parámetros: la distancia de acción ε, como el algoritmo clásico, y una media mínima de ocupación *MinOc*, que sustituye a *MinPts*. Para cada celda *C*, en vez de buscar cierto número de puntos en el radio de acción, se hará la media de ocupación de las celdas que caen dentro de este radio de acción. Si la media supera *MinOc*, se considera *C* como punto (celda) núcleo, y las celdas dentro del radio como puntos alcanzables (que hay que explorar para ver si realmente se tratan de puntos núcleo).

Para no realizar el cálculo de la media en el radio de acción para todas las celdas (sería muy costoso), solo se realizará para aquellas celdas con un mínimo de ocupación. Este mínimo debe ser un número algo inferior a *MinOc* (se usa *MinOc/3*). Por otro lado, se podrá eliminar aún más ruido definiendo un parámetro *PtsCeldas*, que será el umbral de celdas por debajo del cual todo cluster se considerará ruido y se despreciará.

En el Anexo L se incluye un pseudocódigo de este algoritmo.

3.4.3.3 Velocidad de los clusters

Una vez clusterizada la rejilla, se hallan las velocidades de los clusters. Se realiza una media ponderada de las velocidades de las celdas, ponderando por la masa de ocupación de cada celda. Con esto, se logra pasar de velocidades de celdas a velocidades de objetos.

$$vel(cluster) = \frac{1}{\beta} \sum_{C\ in\ cluster} vel(C) * M_{act}^{(C)}(O)$$

Donde:

$$\beta = \sum_{C\ in\ cluster} M_{act}^{(C)}(O)$$

3.4.3.4 Resultados

Así, el DBSCAN suavizado es capaz de identificar objetos y agrupar sus celdas en el mapa de ocupación. Esto ayuda también a eliminar ruido, ya que zonas con poca probabilidad de ocupación no lograrán clusterizarse. Además, se puede inferir la velocidad de estos objetos mediante el método explicado anteriormente. En el Anexo M se muestran un par de ejemplos del clusterizado en situaciones de conducción real.

3.4.4 Algoritmo de asociación

Una vez están extraídos los objetos (clusters) de la rejilla en cada instante, sería interesante ser capaces de asociarlos entre iteraciones sucesivas. Esto es, deducir qué cluster en el instante k+1 correspondía a qué cluster en el instante k. Esto será muy útil. En primer lugar, ayudará a lograr una mejor representación de los clusters. Se podrá dar una sensación de continuidad entre dos mapas de instantes sucesivos usando el mismo color para un cluster y su asociado en el siguiente instante. Así, se podrá seguir la pista visualmente de los objetos en la vía. Por otro lado, también puede ayudar a inferir, a través de la ocupación de la rejilla (en particular de los clusters), la velocidad dl cluster, ya que se tiene el mismo objeto representado en dos instantes de tiempo sucesivos (idóneamente se tendrá el objeto, representado por el cluster, identificado en muchos instantes de tiempo sucesivos, mediante la asociación continuada, como muestra la figura).

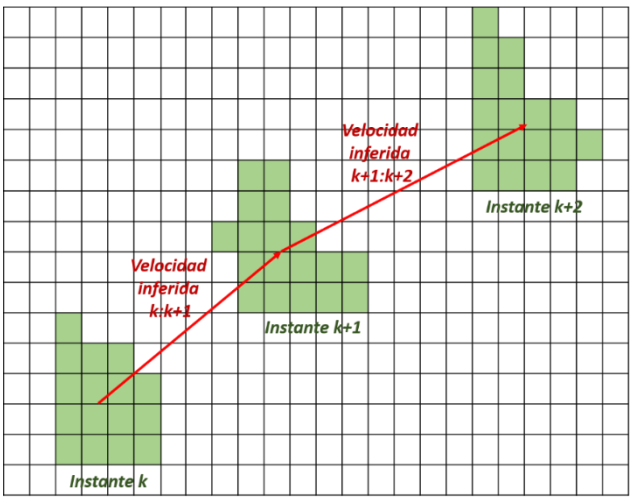

***Figura 3-86**: Ejemplo de asociación continuada de un cluster*

Para calcular el asociado de cierto cluster en un instante anterior, se define el Índice de Incidencia entre dos clusters como una función IDI: (cluster x cluster) ⟶ [0,1]. Es el porcentaje de celdas que tienen en común dos clusters en cierto instante de tiempo. Como entrada, se utiliza un cluster C del instante k y otro C' del instante k+1. Se calcula en tres pasos:

1. Se aplica la traslación y el giro inducidos por el movimiento del coche a los puntos medios de las celdas de C, y el cluster resultante de esta transformación será el conjunto de celdas que contienen estos puntos medios transformados.
2. Se aplica a C su velocidad estimada en el instante k.
3. Una vez realizadas estas dos operaciones sobre C, se tendrá una estimación de la posición de este cluster en el instante k+1 (teniendo en cuenta el nuevo sistema de referencia y la velocidad estimada del cluster). Finalmente, se calcula el porcentaje de celdas que coindiden entre C y C' (que será el índice de incidencia).

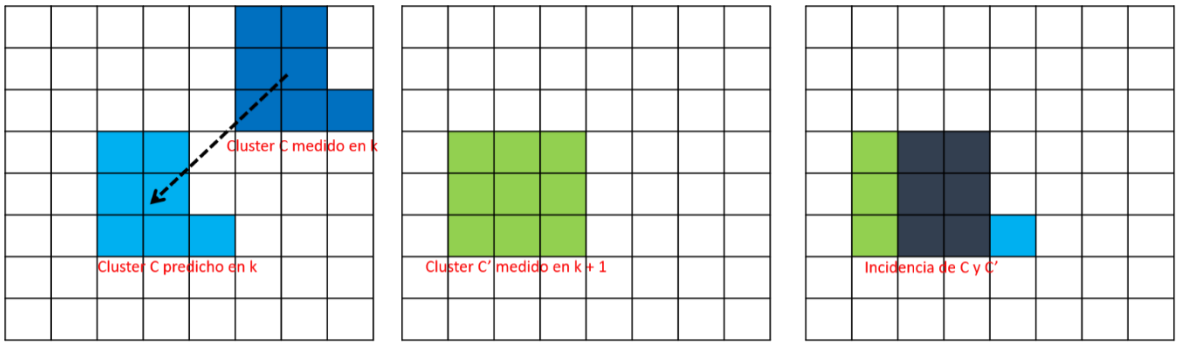

***Figura 3-97**: Representación gráfica de cálculo de IDI entre C y C'.*

$$IDI(C, C') = \frac{2 * \#celdas\ compartidas}{\#celdas\ C + \#celdas\ C'}$$

El IDI podrá interpretarse como la probabilidad de que dos clusters en distintos instantes de tiempo sean el mismo. Se utilizará esta interpretación para asociar los clusters del instante k+1 a los del instante k.

Para ello, se toma una decisión: los clusters sólo podrán asociarse uno a uno. Es decir, ni dos clusters podrán fusionarse en uno, ni un cluster podrá dividirse en dos. Aunque esto en la vida real puede ocurrir, no nos preocupará, pues no conseguir asociar algún cluster en alguna iteración no será demasiado problemático, ya que lo único que ocurrirá es que ese cluster no obtendrá las ventajas de ser asociado (que luego se discutirá en la Sección 3.4.5) en esa iteración. Aun así, siempre podrá asociarse en instantes posteriores. Además, este simple algoritmo garantiza que esta asociación no sea muy costosa, como podría serlo con algún algoritmo más sotisficado.

La asociación se llevará a cabo usando una tabla. Las columnas corresponderán a los clusters del instante anterior y las filas a los de este instante. Como ilustran las tablas, en el instante anterior se clusterizaron 5 objetos (A, B, C, D, E) y en este instante sólo 4 (α, β, γ, δ). Los subíndices sólo indican a que instante pertenece cada cluster. Una vez creada la tabla, mediante el cálculo del IDI entre cada par de clusters, se irán asociando buscando la mejor opción. Esto es, se busca el IDI mayor en la tabla, y se asocian estos dos clusters (primera tabla). Luego se cambia el valor a 0 en esa fila y columna representando que esos dos clusters quedan inasociables (segunda tabla). Se repite la operación (tercera tabla en adelante). Se acaba cuando todos los valores de la tabla sean ceros, habiendo completado la asociación.

A continuación, se muestra una figura que representa una ejecución del algoritmo de asociación. En el instante anterior teníamos 5 clusters (columnas) y en el instante siguiente

4 clusters (filas). En este caso, se han asociado D con **δ**, C con α y B con β. Sin embargo, γ queda sin asociar con un cluster del instante anterior, quizá debido a que es un objeto que entra en el campo de visión.

	A_k	B_k	C_k	D_k	E_k
α_{k+1}	0	0	0.7	0.1	0
β_{k+1}	0	0.3	0	0	0
γ_{k+1}	0	0	0.5	0	0
δ_{k+1}	0	0	0	0.8	0

	A_k	B_k	C_k	D_k	E_k
α_{k+1}	0	0	0.7	0	0
β_{k+1}	0	0.3	0	0	0
γ_{k+1}	0	0	0.5	0	0
δ_{k+1}	0	0	0	0	0

	A_k	B_k	C_k	D_k	E_k
α_{k+1}	0	0	0.7	0	0
β_{k+1}	0	0.3	0	0	0
γ_{k+1}	0	0	0.5	0	0
δ_{k+1}	0	0	0	0	0

	A_k	B_k	C_k	D_k	E_k
α_{k+1}	0	0	0	0	0
β_{k+1}	0	0.3	0	0	0
γ_{k+1}	0	0	0	0	0
δ_{k+1}	0	0	0	0	0

	A_k	B_k	C_k	D_k	E_k
α_{k+1}	0	0	0	0	0
β_{k+1}	0	0.3	0	0	0
γ_{k+1}	0	0	0	0	0
δ_{k+1}	0	0	0	0	0

	A_k	B_k	C_k	D_k	E_k
α_{k+1}	0	0	0	0	0
β_{k+1}	0	0	0	0	0
γ_{k+1}	0	0	0	0	0
δ_{k+1}	0	0	0	0	0

***Figura 3-108**: Ejemplo de ejecución del algoritmo de asociación.*

En resumen, para asociar los clusters se deben calcular los IDIs entre los clusters de los dos instantes en cuestión y luego asociarlos buscando siempre la mejor opción. El lector puede ahora preguntarse si una asociación entre clusters con un IDI muy bajo es adecuado: en la Sección 3.4.5.1 se explica cómo decidir cuáles de estas asociaciones son suficientemente fiables como para ser finalmente efectivas. A continuación, se muestra la primera utilidad de la asociación: mejorar la visualización de los clusters a lo largo del tiempo. Se muestran del mismo color los clusters que se han asociado.

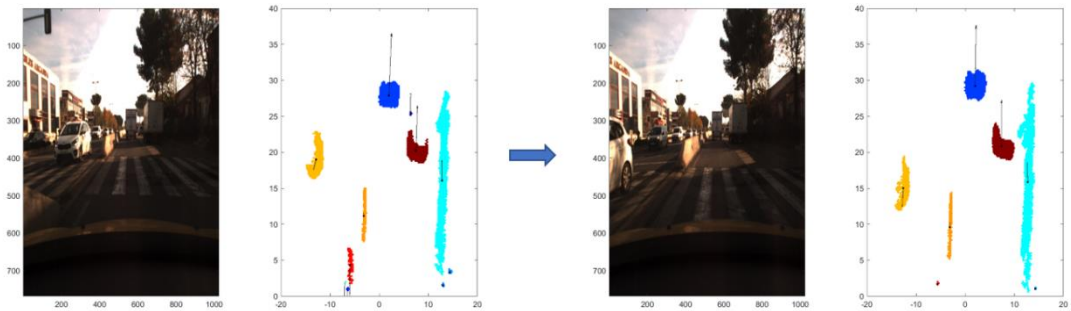

***Figura 3-19**: Asociación de clusters en dos instantes sucesivos.*

3.4.5 Uso del clustering para mejorar el filtro de partículas

Se puede utilizar el clustering para mejorar el filtro de partículas. Aparte de las mejoras evidentes (extracción de objetos en la vía y sus velocidades, eliminación de ruido), se puede ir más allá. El clustering permitirá calcular con criterio la probabilidad de asociación de cada celda (tema que se quedó abierto en la Sección 3.3.3.6) y también mejorará la estimación de velocidades. Pero antes, se debe dar algún paso previo.

3.4.5.1 Cálculo de la confianza de asociación (CA)

Llegados a este punto, ya se sabe cómo asociar los clusters de distintos instantes. Pero, ¿cómo de buena es esa asociación? La confianza de asociación aportará una medida de la calidad de la asociación. Esto es, de la seguridad que se tiene de que esos dos clusters representan el mismo objeto en instantes distintos. Durante la asociación, la posición y la velocidad de los clusters eran determinantes. Ahora lo serán el tamaño y la forma de los mismos.

La idea es similar a la del IDI. En primer lugar, una vez se han asociado satisfactoriamente dos clusters, se calculan sus celdas medias (esto es, calcular el punto medio del cluster, y la celda media será aquella que lo contenga). Luego, se superponen ambos clusters "pegándolos" por la celda media, y se calcula el porcentaje de celdas coincidentes. Este porcentaje nos dará una medida de cuanto realmente se parecen estos dos clusters. Se denota a este porcentaje *p*.

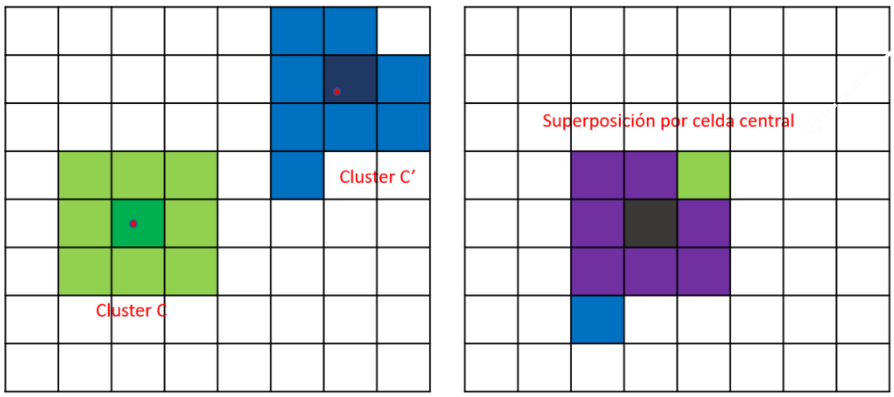

***Figura 3-110**: Representación gráfica del cálculo de p(C, C').*

Una vez calculado *p*, se calcula a partir de este la confianza de asociación. La decisión que se toma es que dependerán linealmente el uno del otro, a partir de cierto *umbral* (se ha tomado la decisión de diseño de fijarlo a 0.7). Es decir, se tiene que la confianza de asociación entre los clusters C y C' es:

$$CA(C, C') = \begin{cases} 0, & p < umbral \\ \dfrac{p - umbral}{1 - umbral}, & p \geq umbral \end{cases}$$

Esta toma valores entre 0 y 1, designando 0 a dos clusters asociados pero que realmente no corresponden al mismo objeto, y 1 a dos clusters asociados con absoluta certeza de que realmente representan el mismo ente.

3.4.5.2 Cálculo de la probabilidad de asociación de una celda

Se recuerda al lector que la probabilidad de asociación de una partícula a su celda mide la probabilidad de que tal partícula esté asociada al momento de la celda en la que se encuentra. Esta celda a su vez puede pertenecer a un cluster o no. Las partículas que nacen en las celdas con baja probabilidad de asociación tenderán a hacerlo con una distribución aleatoria. Sin embargo, una alta probabilidad de asociación hará que las partículas que nazcan en esa celda sean en su mayoría asociadas, y por tanto nacerán con una velocidad similar a la de la celda en la que se encuentran. Esto interesará cuando se esté seguro de que tal celda pertenece a un ente mayor, y que su velocidad es correcta. Esto es, cuando se tenga una CA del cluster superior alta. Se fija pues una probabilidad de asociación mínima *minPa,* y un valor del CA a partir del cual la probabilidad de asociación crecerá linealmente, *umbralCA.* Se podrá calcular la probabilidad de asociación de una celda dinámicamente así:

$$p_A = \begin{cases} minPa, & CA \leq umbralCA \\ minPa + (1 - minPa)\,\dfrac{CA - umbralCA}{1 - umbralCA}, & CA > umbralCA \end{cases}$$

Donde CA es la confianza de asociación del cluster al que pertenece tal celda.

3.4.5.3 Reajuste de velocidades

Aprovechando que se han conseguido asociar los clusters, se puede inferir la velocidad de éstos, y así poder reajustar las velocidades de las partículas en él para que estimen mejor el movimiento en la siguiente iteración.

Lo primero que se debe hacer es inferir la velocidad de los clusters. Se calcula el vector de movimiento de un cluster respecto a su asociado en el instante anterior (vector entre sus centros de gravedad), y luego se divide por el tiempo transcurrido entre ambos instantes.

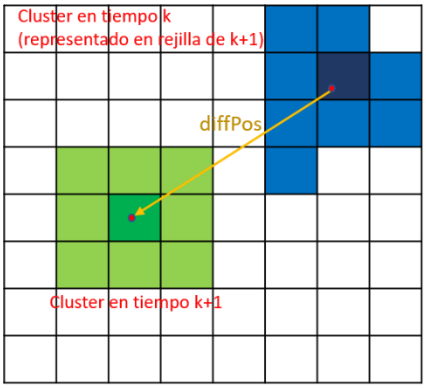

***Figura 3-121**: Representación gráfica del cálculo la diferencia de posición inferida*

La velocidad inferida viene dada por la expresión:

$$vel_{infer} = \frac{diffPos}{T(k, k+1)}$$

Una vez calculada esta velocidad inferida, se puede modificar la distribución de los nacimientos asociados, para que cierto porcentaje de ellos no nazcan con la velocidad de su celda, sino con la velocidad inferida del cluster. El número de partículas que nacerán con la velocidad del cluster será proporcional al CA. De hecho, se toma la decisión de que sea exactamente CA.

Así, un nacimiento asociado tendrá por velocidad la del cluster con probabilidad CA y la de la celda con probabilidad 1 – CA, sumándole cierto ruido aleatorio.

3.4.6 Logros y resultados.

Gracias al clustering, se han logrado los objetivos perseguidos. En primer lugar, se han conseguido distinguir objetos en la vía (en lugar de simple ocupación) y sus velocidades, lo cual es lo más relevante para el coche autónomo a la hora de calcular su trayectoria a seguir. También se tiene un método para eliminar ruido, ya que este no se clusteriza, o se obvian los clusters por ser demasiado pequeños. Además, se ha mejorado el filtro de partículas, calculando dinámicamente la probabilidad de asociación para cada celda, y reajustando velocidades. Esto reduce mucho el tiempo de ejecución, pues ahora menos partículas son necesarias. En algunos escenarios, se ha conseguido reducir el número de partículas de 20000 a 1000 (reducción del 95%) para obtener resultados igual de buenos.

A continuación, se muestra un ejemplo. De izquierda a derecha, tenemos el instante capturado, el clustering sin aplicar reajuste de velocidades y el clustering aplicando reajuste, ambas utilizando 1000 partículas. Se aprecia que el hecho de aplicar el reajuste hace que se calculen correctamente los momentos de los coches, consiguiendo buenos resultados con un número ínfimo de partículas.

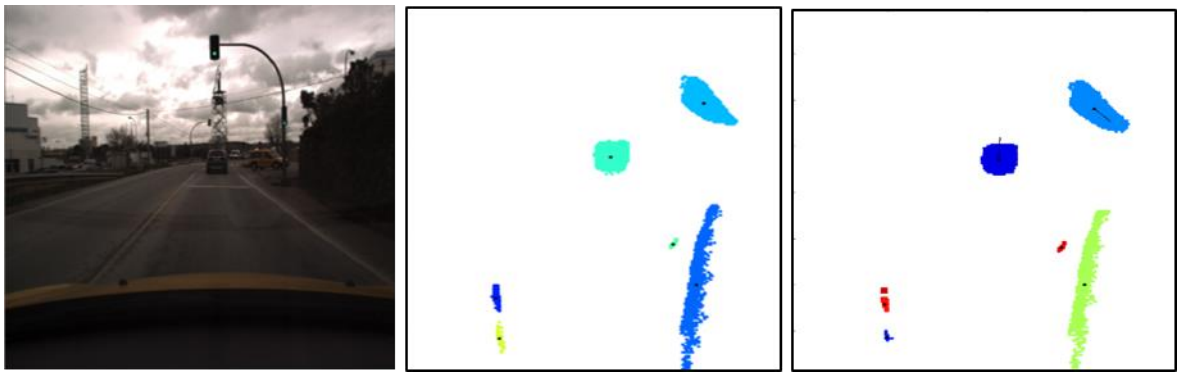

***Figura 3-22**: El reajuste de velocidades mejora el cálculo de estas.*

3.5 Resumen del diseño

Se presenta a continuación un pequeño repaso del esquema general del diseño. El primer problema era calcular una rejilla de ocupación instantánea. Esto se logra mediante un pool de opinión y las distribuciones de ocupación y verosimilitud en función de los impactos recogidos por el Lidar. El segundo paso ha sido implementar el filtro de partículas, que permitió combinar la información presente con la pasada, y de paso calcular las velocidades de las celdas. Este paso ocupa la mayor parte del tiempo de ejecución, y el número (fijo) de partículas será el principal factor que determine tanto este tiempo como la precisión del algoritmo. Por último, se aplica el DBSCAN suavizado para clusterizar la rejilla, extrayendo los objetos de la vía y sus velocidades. También nos sirvió para mejorar algunos aspectos del filtro de partículas. Finalmente, se ha logrado un software que es capaz, a partir de nubes de puntos en distintos instantes y la posición del coche, de generar un mapa dinámico de los objetos de su entorno y sus velocidades.

4 Desarrollo

El software desarrollado está escrito en Matlab. Las razones de elegir este lenguaje son varias. En primer lugar, es un lenguaje sencillo, de más o menos alto nivel, orientado a objetos, y muy legible para cualquier persona con conocimientos de programación. Además, es muy eficiente respecto al uso de matrices, muy utilizadas en el desarrollo de este TFG. Tiene una amplia gama de funciones de representación que son muy útiles para mostrar los resultados. Además, el propio entorno de programación tiene muchas ayudas a la hora de debuguear, lo que facilitó el desarrollo en gran medida. Además, en el CSIC, se tiene la costumbre de probar cualquier código primero en Matlab, y luego, si es viable, transcribirlo a C++ debido a su mayor eficiencia.

El método de desarrollo fue incremental iterativo. Esto es: se iban creando distintos módulos con funcionalidad propia, y luego según se encuentran nuevos problemas o se planteaban nuevos objetivos o requisitos, se iban añadiendo nuevos módulos. Lo primero que se hizo fue estudiar papers con descripciones de distintos tipos de filtros de partículas, y se acabó optando por la aproximación propuesta por Damlier and Ulm [6]. También se tomaron algunas ideas del algoritmo CMCDOT [5] propuesto por el grupo de investigación francés INRIA. Así, se implementó una primera versión del filtro de partículas. Para ello, se utilizó un software desarrollado por un antiguo trabajador del centro, Víctor Alonso Mendieta, que transformaba una nube de puntos en un mapa de ocupación básico. Tras realizarse las primeras pruebas, se observó que el software de Víctor no servía, debido a que los mapas generados eran demasiado simples. Así, se desarrolló un software propio para la detección instantánea, basado en el Pool de opinión [8]. Esta aproximación generaba demasiado ruido, y se tuvieron que ingeniar métodos para reducirlo (Apéndices F, G y H). Una vez llegados a este punto, se tenía un filtro de partículas completamente funcional, pero no manera de comprobar el correcto funcionamiento de los cálculos de velocidad, por ser a nivel celdilla. Así, se propuso realizar un clustering simple para representar velocidades a nivel objeto. Tras una buena primera experiencia con DBSCAN, este se retocó, resultando en DBSCAN suavizado, cuyos resultados fueron aún mejores. Más tarde se descubrieron otras ventajas que el clusterizado podía ofrecer, como por ejemplo mejorar el filtro de partículas inicial. Por último, se realizaron una serie de pruebas con datasets de distintos escenarios para corregir errores y asegurarse que el software funcionaba bien en diversas situaciones.

Durante todo el desarrollo, siempre se han tenido en cuenta una serie de aspectos importantes. En primer lugar, que el código en algún punto debería funcionar en tiempo real. Por ello, se ha tratado de reducir al máximo el tiempo de ejecución, buscando la mayor eficiencia a la hora de programar. Además, la mayor parte del código se ha implementado para que sea paralelizable, y en particular las partes que requieren mayor esfuerzo computacional (tratamiento de celdas y de partículas). Se pretendía así que, al transcribir el código a otro lenguaje, en busca de la máxima eficiencia, se pudiese realizar directamente sin cambiar la lógica del mismo. Por último, se ha procurado que el código sea modular, esté

organizado por funcionalidad y sea legible. La cantidad de comentarios es abundante, ayudando a la comprensión del código.

En el Anexo C, se comentan a rasgos generales la estructura de ficheros del software, además del recorrido temporal de una ejecución normal del mismo.

5 Integración, pruebas y resultados

5.1 Integración

Lamentablemente, el software desarrollado en este proyecto no ha sido aún integrado en el coche. Las razones son varias. En primer lugar, la totalidad del código integrado en el coche autónomo está escrito en C++, luego es conveniente que todo nuevo código siga la misma línea. Por otro lado, este software aún está en versión experimental. Aunque se ha probado en diversos escenarios, aún le queda enfrentarse a muchos otros, debido a la amplísima gama de situaciones no contempladas en este trabajo (como túneles o condiciones climatológicas adversas). Por último, este no funciona en tiempo real, con lo que sería inútil tratar de hacerlo funcionar en la vía.

Por todos estos motivos, el único entorno en el que se ha ejecutado el código ha sido en una computadora proporcionada por el CSIC, utilizando datasets pregrabados. Todas las pruebas y resultados obtenidos han sido llevados a cabo en estas circunstancias.

5.2 Pruebas

La mayoría de las pruebas han sido de debugueo y experimentales. Debido a que este ha sido un trabajo de investigación, muchos de los resultados (especialmente aquellos algoritmos originales) han sido fruto de la prueba y del error. Así, tras desarrollar un módulo, inmediatamente debía probarse: no sólo su correcto funcionamiento, sino también su eficiencia. En función de los resultados, se planteaba aceptar ese módulo, tratar de mejorarlo pensando en una nueva implementación o en algún reajuste, o descartar la idea debido al mal funcionamiento en los datasets o simplemente por no ser rentable el cómputo necesario a cambio de las ventajas que ofrecía.

Una vez desarrollado el software, mis compañeros del CSIC Antonio Artuñedo y Jorge Godoy me ayudaron a grabar una amplia gama de escenarios. Desde algunos controlados en el propio recinto del CSIC, hasta otros más reales en las carreteras y calles abiertas al tráfico. Esto fue de gran ayuda para mejorar el software. Al principio, se contaba con un solo dataset, y el código sólo se orientaba a que procesase este lo mejor posible. Sin embargo, los escenarios reales son infinitos, y lo que funciona bien en un escenario puede funcionar menos bien en otros. No es lo mismo la detección de objetos estáticos o dinámicos, hacerlo con el coche en movimiento o en escenarios con muchos objetos o pocos. En el Anexo N se listan los tipos de datasets utilizados.

Así, se realizaron pruebas para tratar de ajustar los parámetros lo mejor posible para que el algoritmo funcionase correctamente en los distintos escenarios para los que se tenían datasets. El problema es que decidir qué parámetros funcionan mejor es algo muy relativo.

No hay forma de medir cuán bien funciona el algoritmo, pues esto depende de muchos factores: el escenario sobre el que se aplica, tiempo de ejecución, estimación de probabilidad de ocupación, los clusters y sus velocidades, cantidad de ruido, etc. Cómo validar las ejecuciones para así tratar de buscar las mejores combinaciones de parámetros podría ser el foco para otro TFG por sí solo (ver Sección 6.2.1).

Aunque el objetivo del TFG no era encontrar la manera óptima de implementar la rejilla de ocupación, sino más bien comprobar que la aproximación es viable, durante algunas semanas se realizaron una serie de pruebas para tratar de buscar una buena combinación de parámetros. Para ello, se recogían datasets relevantes para probar cierto parámetro. Luego, se ejecutaba el algoritmo fijando el resto de parámetros, pero variando el valor del parámetro a ajustar. Por último, se puntuaban una serie de aspectos como la probabilidad de ocupación estimada, el ruido generado, la velocidad estimada o el clusterizado, y se escogía el valor del parámetro con mejor valoración.

Estas pruebas se pueden complicar bastante pues muchos parámetros son muy dependientes de otros. Por ejemplo, el radio y la media de ocupación del algoritmo DBSCAN suavizado. Estos valores tienen más sentido estudiarlos a la vez, lo que llevaría a realizar una doble variación de parámetros (lo que antes eran n ejecuciones, ahora serían n^2). Esto se complica aún más cuando se cuenta con extensos subconjuntos de parámetros altamente correlacionados. Por ello, este ajuste de parámetros es muy complejo. En el Anexo O se incluye un ejemplo de un ajuste paramétrico que se llevó a cabo.

5.3 Resultados finales del software

Esta sección vendrá dada de manera externa pues para representar los resultados que se obtienen se necesita algún tipo de animación o vídeo. Adjunto a esta memoria una presentación de Powerpoint con una serie de animaciones que muestran ejemplos de resultados obtenidos.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Combinar hardware y software para tratar de imitar digitalmente algo tan complejo como el ojo humano y su conexión con el cerebro no es trivial. De hecho, es una rama que está aún en pleno desarrollo, y a la que le queda mucho por recorrer. La conducción autónoma depende fuertemente de los avances en percepción artificial, y en especial de la detección de obstáculos, parte crucial de esta. Aunque aún se está muy lejos de lograr comercializar automóviles completamente autónomos, desde el Programa Autopía del CSIC (desarrollado en la sede de Arganda del Rey del Centro de Automática y Robótica) se están haciendo grandes esfuerzos para que este momento llegue lo antes posible. Los expertos están convencidos de que en un tiempo no muy lejano, algunas actividades del ser humano como conducir serán delegadas a las máquinas. Éstas serán capaces de llevarlas a cabo de manera más segura que las personas, ya que tienen más capacidad de cálculo, y no se ven afectadas por estados de ánimo, enfermedades, insomnio o embriaguez, causantes de una gran mayoría de los accidentes al volante.

Este software puede aportar mucho a la conducción autónoma. Aunque le quedan muchas pruebas por pasar (por haber miles de situaciones distintas en carretera), los resultados obtenidos en los datasets grabados son prometedores. Desde luego, valdrá la pena continuar con esta línea de investigación, pues se ha probado que es una aproximación más que viable al problema de la detección de obstáculos.

Confío en que mi paso por Autopía sea relevante para ellos y que este trabajo les sea de utilidad, al igual que ha sido muy educativo para mí mi paso por allí. He aprendido que el mundo de la investigación es complicado, pues hay que saber detectar -o al menos intuir- qué son buenas ideas y tienen gran probabilidad de éxito frente a aproximaciones que no llevan a ninguna parte. Esta cualidad tan complicada de adquirir es indispensable para un investigador prolífico y eficiente.

He adquirido muchos conocimientos mientras desarrollaba este trabajo. En primer lugar, es el primer proyecto serio en el que no tengo unas pautas guiadas, sino que tenía libertad para explorar e investigar a mi aire. Aunque esto añade un grado de dificultad, pues en ningún momento se está seguro de que lo que se está haciendo es óptimo o siquiera si va a funcionar, resulta muy gratificante cuando creas algo nuevo y obtienes buenos resultados. Por otro lado, el hecho de no tener una vía de aprendizaje única, hace que sea importante la lectura crítica, para así poder valorar las distintas fuentes de información y guiarte por aquella que consideres más apropiada. Estas cualidades no se enseñan en la escuela ni en la universidad, pero considero que son muy importantes para la vida laboral y que deberían tener cabida en alguna asignatura además del TFG.

6.2 Trabajo futuro

A continuación, se exponen una serie de ideas para continuar con la línea de investigación de este proyecto. También se porpone alguna idea para tratar de mejorar el software que se ha desarrollado.

6.2.1 Búsqueda de parametrización óptima.

Como ya se ha indicado en la Sección 5.2, encontrar la parametrización óptima del algoritmo no es para nada trivial. Aunque se ha hecho lo posible para hallar una buena parametrización, es posible que haya algunas combinaciones aún por descubrir que proporcionen el máximo rendimiento posible. El hecho de que esta parametrización óptima pueda además depender del entorno hace aún más complejo este problema.

Lo primero que se debería hacer es definir algún tipo de heurística para decidir cuándo una ejecución es mejor que otra. Esto es, tratar de definir alguna medida de lo bien que se modela la ocupación del espacio, el clusterizado y las velocidades. Para lograrlo, quizás haya que grabar una serie de datasets muy controlados, dónde todas las distancias, tiempos y velocidades de los objetos estén registrados de una manera muy precisa. Una vez se cuente con esto, podrán compararse estas cantidades con las inferidas por el software. Al no ser un algoritmo determinista, habrá que repetir las ejecuciones varias veces para reducir el factor aleatorio.

Una vez la máquina pueda calcular lo precisa que ha sido cierta ejecución, sería conveniente aplicar alguna técnica de inteligencia artificial (quizás algún algoritmo genético o red neuronal) para ir variando estos parámetros y tratar de alcanzar la combinación óptima. No sólo esto, sino que la propia máquina fuese capaz de variar estos parámetros en función del entorno en el que se encuentre. Por ejemplo, variar el rango de visión del algoritmo en función de si está en una ciudad o en carretera, o simplemente extraer de alguna manera la velocidad máxima de la vía y modificar este parámetro en el algoritmo.

6.2.2 Tracking. Uso de inteligencia artificial para distinguir obstáculos.

Una técnica muy utilizada en la conducción autónoma es el tracking. Este es similar a la asociación de clusters descrita en la Sección 3.4.4, pero teniendo en cuenta no solo el instante anterior, sino todos los instantes anteriores, y usando además un modelo de predicción. Combinando la información de todos estos instantes se lograría un mejor reajuste de las velocidades, además de mantener la continuidad de los clusters aunque en una iteración no se consigan asociar correctamente.

Al realizar un seguimiento de los objetos (clusters), se podrían extraer también otros datos también muy valiosos de manera más precisa, como la forma del objeto, su tamaño o su velocidad media. Usando estas variables, y quizás alguna técnica de inteligencia artificial (árboles de decisión quizás funcionasen bien) se podría deducir la naturaleza de estos obstáculos. Así, sería posible distinguir coches de personas o de quitamiedos, que sin duda ayudaria a la hora de planificar rutas o tratar de predecir movimientos.

6.2.3 Mejoras del clustering.

El algoritmo de clustering propuesto es aún algo básico porque sólo se basa en la cercanía de las celdas y no en otros parámetros. Podría usarse también la velocidad de las celdas como métrica añadida al DBSCAN suavizado. Así, dos objetos que colisionan podrían distinguirse en todo momento, cosa que no es posible con la implementación actual.

6.2.4 Cálculo dinámico de la vía.

Una posibilidad muy interesante sería calcular dinámicamente los límites de la vía. Con ello, se conseguiría ignorar partes del campo de visión del algoritmo que no aportan nada, pero que consumen procesado (ya que se asignan partículas a la masa en estos lugares). Se lograría una mejora en la eficiencia del algoritmo, logrando que el estimador de ocupación se centre solamente en las zonas con cierto interés. Esto no es extremadamente complejo ya que los investigadores del CSIC utilizan una aproximación similar para los cálculos de trayectorias del vehículo autónomo. Para ello, usan unos mapas abiertos y con acceso libre a las vías de España (Open Street Maps).

6.2.5 Integración en el coche.

Este es quizás el trabajo futuro más interesante. Para ello, primero habrá que conseguir que el software funcione en tiempo real, manteniendo la precisión. Un primer paso sería transcribir el código a un lenguaje más eficiente, como podría ser C++. Además, una implementación paralela reduciría mucho el tiempo de ejecución (los algoritmos están pensados para que sean fácilmente paralelizables, el problema es que en Matlab no se puede especificar este paralelismo). Por último, es necesaria la utilización de una máquina mucho más potente que el portátil suministrado por el CSIC en el que ejecutaba mis pruebas, el cual cuenta con solamente dos procesadores. La utilización de una tarjeta gráfica podría ser una buena idea. De hecho, Autopia está ahora mismo en proceso de integración de una plataforma embebida de altas prestaciones basada en GPUs (Nvidia Drive PX2).

Con todos estos arreglos, confío en que se pueda lograr que el software se ejecute en tiempo real. Sólo faltaría integrarlo en el software ya existente del automóvil autónomo.

Referencias

[1] Varios autores, «Autopia Program» [En línea]. Available: https://autopia.car.upm-csic.es/index.php.

[2] H. P. Moravec, «Sensor Fusion in Certainty Grids for Mobile Robots» *Al Mag*, vol. 9, pp. 61-74, 1988.

[3] A. Elfes, «Using occupancy grids for mobile robot perception and navigation» *Computer*, nº 22, pp. 46-57, 1989.

[4] Cristopher Coué, «Using Bayesian Programming for Multi-Sensor Multi-Target Tracking in Automotive Applications» *IEEE International Conference on Robotics and Automation* , nº 14, pp. 2104-2109, 2003.

[5] Lukas Rummelhard, «Conditional Monte Carlo Dense Occupancy Tracker» *HAL*, vol. 1, pp. 1-7, 2015.

[6] Dominik Nuss, «A Random Finite Set Approach for Dynamic Occupancy Grid Maps with Real-Time Application» *The International Journal of Robotics Research*, nº 1, pp. 1-21, 2016.

[7] Sascha Steyer, «Grid-Based Envioronment Estimation Using Evidential Mapping and Particle Tracking» *IEEE Transactions on Intelligent Vehicles*, vol. 3, nº 3, pp. 5-20, 2018.

[8] Juan David Adarve, «Computing Occupancy Grids from Multiple Sensors using Linear Opinion Tools» *HAL*, vol. 1, pp. 1-13, 2012.

[9] Jean Dezert, «On the mathematical theory of evidence and Dempster's rule of combination» *HAL*, nº 1, pp. 1-12, 2011.

Glosario

Paper[1]	Palabra inglesa que equivale a un ensayo científico de poca extensión y que trata un tema muy específico.
Visión[2]	La visión por computador es una disciplina de la informática que trata de extraer y analizar la información de interés contenida en una imagen o secuencia de imágenes. Los campos de aplicación crecen cada día y van desde el reconocimiento de caras al diagnóstico precoz de enfermedades, pasando por la detección y localización de objetos.
Filtro de partículas[3]	Posible implementación de una rejilla de ocupación. Consiste en la inicialización de miles de partículas que simulan la masa del entorno, y que se mueven por el espacio debido a su velocidad.
Tiempo real[4]	Dícese del software que debe operar a la misma velocidad que los eventos que ocurren a su alrededor, dando respuesta inmediata o casi inmediata a los mismos.
Paralelizable[5]	Dícese de aquel código que permite ser ejecutado por distintos procesadores que trabajan al mismo tiempo, cada uno realizando una parte del trabajo, para luego combinar los resultados y llegar al resultado final. Si se cuenta con el hardware necesario (esto es, una máquina con suficientes procesadores), esta cualidad es fundamental para maximizar la eficiencia del código.
Clustering[6]	Técnica que consiste en agrupar puntos del espacio por cercanía u otros criterios.
Debugueo[7]	Acción de buscar bugs (errores en el software).
ADAS[8]	Sistemas de asistencia a la conducción, que incrementan notablemente la seguridad activa y suponen los primeros pasos hacia una conducción completamente autónoma. Incluyen desde el frenado autónomo de emergencia con detección de peatones, la detección de ángulo muerto o el sistema de detección de fatiga, a la alerta de cambio involuntario y de carril, el mantenimiento activo en el carril, la alerta de tráfico trasero cruzado o el reconocimiento de señales de tráfico
Rejilla[9]	Simulación informática del entorno, mediante la proyección 2D del mismo, y dividiendo el espacio en celdas diminutas.
Nube de puntos[10]	Conjunto de impactos detectados por un láser en un barrido.
LIDAR[11]	Dispositivo que permite determinar la distancia desde un emisor láser a un objeto o superficie utilizando un haz láser pulsado. La distancia al objeto se determina midiendo el tiempo de retraso entre la emisión del pulso y su detección a través de la señal reflejada.

Falso impacto[12]	Error de medida de un láser, consistente en detectar masa allí dónde no la hay (o dónde no se quiere detectarla)
Resolución angular[13]	En una rejilla polar, el conjunto de celdas que barren la misma sección angular. Si se trata a la rejilla polar como una matriz, correspondería a una columna de esta.
Dataset[14]	Palabra inglesa que significa conjunto de datos. En nuestro caso, serán grabaciones de imágenes, GPS y Lidar del coche durante cierto intervalo de tiempo, que luego utilizaré para testear el algoritmo offline.
Pitch[15]	Fenómeno en la conducción en el que la inclinación del coche varía constantemente debido a irregularidades o cambios de curvatura en la carretera, o la propia vibración del coche inducida por el motor.

Anexos

A Notación

En este anexo, se aclaran aspectos relativos a la notación. Aunque el significado de la notación está explicado a lo largo de la memoria, ésta en ocasiones puede ser densa y por ello se ha considerado necesario añadir este anexo para poderse consultar en caso de duda.

- $\boldsymbol{P(C|Y_i)}$**:** Probabilidad de ocupación de la celda C calculada por el sensor (la capa) Y_i.
- $\boldsymbol{w_{i,inc}(C)}$: Verosimilitud de la medida de la celda C calculada por el sensor (la capa) Y_i. El segundo subíndice se refiere al tipo de verosimilitud (si no se especifica, es la verosimilitud global).
- $\hat{\mathrm{C}}(\mathbf{C})$**:** Certidumbre de la medida en la celda C.
- $\boldsymbol{M^{C}_{pred,\,k}(O)}$**:** Masa de evidencia. El primer subíndice denota el tipo de masa de evidencia, y el segundo el instante de la muestra. El superíndice indica la celda en la que se mide, y la O denota ocupada (podría ser L de libre o OL de ocupada y libre).
- $\boldsymbol{x_k^{part}}$**:** Estado de la partícula $part$ en el instante k.
- $\boldsymbol{w^{part}_{pred,k}}$**:** Peso de la partícula $part$ en el instante k. El primer superíndice indica el tipo (en este caso, peso predicho). En caso de faltar este subíndice, indica peso estándar.
- $\boldsymbol{\varrho}^{C}_{b,k}$: Masa recién nacida (para indicar masa persistente se usa p en vez de b como primer subíndice) de la celda C en el instante k.
- $\boldsymbol{v}^{C}_{A,k}$: Partículas asociadas (no asociadas se indica con $\neg A$) que nacen en la celda C en el instante k. No confundir esta notación con $\boldsymbol{v}$ que es el número (fijo) de partículas y $\boldsymbol{v_b}$ que es el número de partículas que nacen a cada iteración.

Nota: En ocasiones faltará algún índice, debido a que éste se sobreentiende y para no sobrecargar la notación.

B Manual del usuario

En este anexo, se detallan las instrucciones para realizar una ejecución del software sobre un dataset pregrabado.

Lo primero será generar los ficheros *.mat* del Lidar y del LCM que servirán como entrada a esta ejecución. Para ello, posiciónese en la carpeta datos. El dataset a ejecutar, que será una carpeta cuyo nombre identificará al propio dataset, debe contener los siguientes ficheros:

- Fichero *lcm.00* que contiene los datos generados por el LCM.
- Carpeta *imágenes* con las imágenes de la cámara. Los nombres de estas seguirán el formato *rectificada_l_sxxxxxxxxx_nsyyyyyyyyy* donde las *x* denotan el segundo en el que fue tomada la imagen y las *y* el nanosegundo.
- Carpeta *datos* con el fichero *puntos.bin* en su interior, conteniendo los datos del Lidar.

Una vez se tiene el dataset correctamente estructurado, se llama a las funciones *CargaLidar* y *CargaLCM* desde la carpeta datos, pasando como argumento el nombre del dataset, y generándose los ficheros del LCM y del Lidar correspondientes con la extensión requerida.

El siguiente paso será instanciar las decisiones de diseño. Para ello, diríjase al fichero *decisiones_disenio.mat* situado en la carpeta principal y cambie las constantes a su antojo. Es importante que la constante *nombre_dataset* sea efectivamente el nombre del dataset a ejecutar. El significado de las constantes está detallado en el Apéndice C. Una vez hecho esto, instancie el objeto contenedor de los parámetros tecleando *d = decisiones_disenio()*.

Por último, ejecute la función *ejecucion(d, ini, fin, nombreCarpeta)* desde la carpeta principal, donde *d* es la instanciación de los parámetros seleccionados, *ini* y *fin* representan las iteraciones de inicio y de fin de la ejecución (siendo cada iteración un barrido del Lidar) y el nombre de la carpeta de resultados vendrá dado por *nombreCarpeta*. En esta carpeta se guardarán los siguientes elementos:

- Fichero *clusters.txt*: Detalla información de los clusters calculados a cada iteración, además de la variación de posición y de ángulo a cada iteración.
- Fichero *decisiones.mat*: Contiene los valores de los parámetros utilizados en dicha ejecución.
- Ficheros *m_xxx.png*: Son los resultados de cada iteración del código. De izquierda a derecha y de arriba abajo, son la imagen del instante, la masa de ocupación predicha, la probabilidad de ocupación actualizada, la distribución de las partículas, los clusters con representación de velocidades con código de colores y los clusters con representación de velocidades vectorial.
- Carpeta *extra*: De generarse, contiene información extra de la ejecución. Para modificar qué se muestra en estos ficheros, modificarlo en el fichero *representación.mat*, colocado en la carpeta Utilidades.

De estar a 1 el flag *guardar_detección*, se genera una carpeta con los resultados de la fase de detección instantánea a cada iteración.

C Estructura del software

En esta sección, se da una visión general de la estructura del software desarrollado. Está estructurado en 6 carpetas (cluster, datos, detección, filtro, mains, utilidades) y un fichero suelto que corresponde a las decisiones de diseño. Se da una visión general de los ficheros de cada directorio así como algún detalle ocasional de implementación. Es importante recalcar que todos los objetos tienen la etiqueta *handle*, que hará que, al pasarse como argumentos de funciones, estos se vean modificados fuera de ellas.

1. Cluster

Esta carpeta contiene la funcionalidad del clustering. Los ficheros que contiene son los siguientes:

- **Cluster.m**: Este fichero contiene la clase cluster. Tiene una serie de propiedades que definen el cluster, como su velocidad, su centro, las celdas que contiene, el id del cluster asociado del instante anterior (de haberlo) y su porcentaje de coincidencia con el mismo, la velocidad inferida calculada y el color con el que lo represento. En cuanto a sus métodos, son principalmente un constructor, la función *calcula_velocidad* que computa el momento del cluster a partir de los de sus celdas, y la función imprimir, que imprime la información del cluster en un fichero proporcionado.
- **Clusterizar.m**: Aplica DBSCAN suavizado a un mapa proporcionado por argumento. Cuenta con las funciones auxiliares: *indices_rejilla_cercanos*, que calcula los índices de las celdas a una distancia menor que la proporcionada; *agrupar_celdas*, que aplica el algoritmo al mapa (pseudocódigo en el anexo E) devolviendo los clusters y sus velocidades; y otras funciones utilizadas para rellenar los campos del cluster como su centro de gravedad o su identificador. Despreciará los clusters demasiado pequeños (eliminando ruido) y devolverá la lista de clusters calculados.
- **Inferir_correspondencias**: Asocia clusters de esta iteración con los de la iteración pasada. Para ello necesita las variaciones de ángulo y posición (calculando así la posición de los clusters de la iteración anterior en este nuevo sistema de referencia), así como el tiempo transcurrido (para aplicarles la velocidad calculada anteriormente). Así, calculará los IDIs entre todos los clusters (algunos los establecerá a cero antes de empezar si la distancia entre los centros de los clusters es demasiado grande), y luego los asociará como se explica en la Sección 3.4.4**.** Establece las asociaciones en las instancias cluster como tal.
- **Reajustar_velocidades**: Calcula el coeficiente de asociación entre los clusters asociados, y si supera cierto umbral, calculará la nueva probabilidad de asociación de las celdas correspondientes y la velocidad inferida del cluster. Se ayuda de la función auxiliar *calcula_peso_cluster*, que calcula el porcentaje de incidencia como en la Sección 3.4.5.1.

2. Datos

En esta carpeta, se guardan los archivos de datos utilizados por el software. Esto es, los ficheros de datos extraídos del Lidar y del LCM (guardados con extensión *.mat*). También se guarda en él carpetas con las imágenes y ficheros *.bin* de cada dataset que se utiliza como prueba. Solo hay dos ficheros que contienen código:

- **CargaLCM.m**: Crea un fichero *.mat* a partir del nombre de un dataset que se le pasa como entrada. El fichero *.mat* resultante contendrá toda la información que genera el LCM, incluyendo la posición en coordenadas del coche, la orientación, la velocidad y la velocidad angular, el giro del volante, y el instante de toma de datos. Este código fue suministrado por la gente del CSIC.
- **CargaLidar.m**: Carga la nube de puntos del Lidar, conteniendo información de cada punto, como la capa que lo detectó, el ángulo y la distancia. También contiene el tiempo del barrido que generó la nube de puntos. Al igual que el fichero anterior, fue suministrado por la gente del CSIC.

3. Detección

Aquí se guardan los ficheros que tienen que ver con la detección instantánea.

- **Combina_capas.m:** Esta función toma como argumento la iteración que se quiere procesar, y devuelve el mapa de ocupación instantánea. En primer lugar, carga la nube de puntos y se queda solo con aquellos que se encuentran en los límites de mi mapa. A los puntos restantes, se les realiza un filtrado de la nube de puntos para eliminar ruido (Apéndice G), colocando flags a los puntos considerados como falsos positivos. El siguiente paso es calcular dinámicamente la distancia al suelo de cada capa (Apéndice F), y utilizar el fichero *w_inc.m* para calcular la verosimilitud angular. A continuación, se computa el mapa de ocupación aplicando gaussianas alrededor de los puntos de impacto (los verdaderos positivos) como se vio en la Sección 3.2, y luego se invoca a *w_choque.m* para calcular la verosimilitud de impacto. Por último, se calculan las probabilidades finales usando el pool de opinión, así como la certidumbre, la masa ocupada y la masa libre, y se transforma el mapa a cartesianas. Hay un módulo extra final que permite imprimir algunos resultados intermedios. Se devuelven los mapas de masa ocupada y libre, además del tiempo en el que se tomó la nube de puntos.
- **W_choque.m**: recibe el número de celdas en distancia, la desviación de la normal de caída gaussiana, el array de índices de impactos y su naturaleza (además de un factor lambda para la desviación de la gaussiana de caída) y devuelve la función de verosimilitud de impacto evaluada en una serie de puntos equidistantes (tantos como indique el valor suministrado en la entrada de la función).
- **W_inc.m**: recibe la distancia práctica del suelo (calculada dinámicamente) así como el número de celdas en distancia, y devuelve la función discretizada de la verosimilitud de inclinación, en una matriz con n filas siendo n el número de capas. Utiliza funciones auxiliares que representan las curvas (en este caso rectas) de verosimilitud de las capas que chocan completamente y parcialmente.

4. Filtro

Esta carpeta contiene una serie de ficheros de métodos auxiliares para el filtro de partículas. También contiene 4 ficheros correspondientes a cuatro clases utilizadas a lo largo de todo el software.

- **Celda.m**: Contiene la clase celda, que recoge la información de una celdilla de la rejilla. Algunos de sus atributos son: los índices de las partículas que contiene, el número de partículas totales, de partículas recién nacidas y las que son asociadas, los distintos tipos de masa (observada, predicha y actualizada) libre y ocupada, la velocidad, el peso de las partículas que nacen en ella, la probabilidad de asociación o el id del cluster al que pertenece, de pertenecer a alguno. Sus métodos más relevantes se usan para para tomar los índices de una celda en la matriz mapa, calcular su velocidad, añadir una partícula a las partículas que contiene o calcular los pesos de las partículas recién nacidas asociadas y no asociadas.
- **Lidar.m**: Clase que simula un barrido de un lidar. Así, sus propiedades son los mapas de masa ocupada y libre (observada) y el tiempo en el que se toma tal muestra. Para generar estos mapas, invocará al método *combina_capas.m*.
- **Mapa.m**: Contiene la información del espacio observable del coche. Esto incluye su ancho y su largo, el número de celdas en horizontal y en vertical, su origen, y una matriz de celdas que representa la rejilla. El constructor establecerá estos valores e inicializará las celdas. Tiene métodos para hallar en qué celda está cierta partícula, o a qué celda pertenecen ciertas coordenadas. *Actualizar_masa* es un método muy importante, pues realiza la combinación de las masas predicha y observada usando las reglas de Dempster-Shafer. *Inicializar_particulas* inicializa las partículas nuevas a cada iteración. Otros métodos calculan las velocidades de las celdas del mapa o representan el mapa de ocupación (la rejilla).
- **Particula.m**: Representa una partícula del filtro. Contiene información de su peso, velocidad y posición. También guarda la celda a la que pertenece, y un flag que indica si es considerada una partícula estática o no. Además de setters, el método *inicializar_partícula_nueva* invoca a los métodos que definen las distribuciones de probabilidad de las partículas que nacen, y les da valores de peso, posición y velocidad. También tiene un método *copy* para copiar una partícula, utilizado en el remuestreo.
- **Actualizar_pesos_persistentes.m**: Método que actualiza los pesos de las partículas persistentes, eliminando aquellas partículas con peso cero. Además, separa las partículas en estáticas y dinámicas, y calcula un array de pesos acumulados de las partículas dinámicas que luego se utilizará a la hora de remuestrear.
- **Crea_particulas.m**: Crea un array de v_b partículas nuevas, estableciéndoles la celda a la que pertenecerán.
- **Inicializar_algoritmo.m**: Inicializa el algoritmo creando un mapa inicial utilizando solamente la masa observada en cierto instante. Para ello, inicializa el mapa y establece las masas actualizadas al valor de la masa observada (pues al inicializar no se tiene otra información). Luego se generan v partículas distribuidas en las celdas proporcionalmente a la masa observada en estas, y con la distribución de probabilidad de un nacimiento no asociado (para su velocidad y su posición dentro de la celda).
- **Movimiento_particulas_y_coche.m**: Simula el movimiento de las partículas. Lo primero que hará, será crear la nueva rejilla invocando a *movimiento_rejilla*. Luego,

calcula la nueva posición de las partículas compensando el movimiento del coche, y les aplica un movimiento continuo uniforme con cierto ruido. Aglomerará las partículas estáticas que caigan en la misma celda.

- **Movimiento_rejilla.m**: Transforma la rejilla de ocupación compensando el movimiento del coche, y asociándole valores a las celdas como se observó en la Sección 3.3.3.1. También calcula la posición de las partículas en el nuevo eje de coordenadas.
- **Ordena_particulas_por_celda.m**: Como su propio nombre indica, ordena el array de partículas por celda, recorriendo la rejilla y viendo qué partículas pertenecen a cada celda. Esto será útil para que las celdas puedan guardar eficientemente qué partículas contienen, tan solo guardando el índice de inicio y de fin de la parte del array de partículas que pertenecen a ellas. Por otro lado, calcula de paso la masa predicha ocupada de cada celda sumando sus pesos, y normaliza sus pesos si se da el caso de que la masa supera el valor uno.
- **Remuestreo.m**: Remuestrea las partículas tal cual se describe en la Sección 3.3.3.7.

5. Mains

En esta carpeta se guardan los ficheros que a menudo se ejecutarán en la barra de comandos de Matlab para generar gráficas resultantes de aplicar el algoritmo sobre cierto dataset.

- **Ejecucion.m**: Esta función ejecuta el código sobre un dataset entre dos índices de barridos del Lidar, guardando la información en una carpeta pasada como argumento. En primer lugar, inicializa todos los ficheros que se generarán. Estos son *decisiones.mat* (contiene las decisiones de diseño utilizadas en esta ejecución), *clusters.txt* (información sobre los clusters calculados en cada iteración), una serie de gráficas correspondientes a cada iteración (que incluye la imagen de la cámara, el mapa de la masa ocupada actualizada, el mapa de la probabilidad de ocupación, la distribución de las partículas, y los clusters representando las velocidades por un código de colores y por vectores) y en ocasiones una carpeta extra con gráficas opcionales. Invoca a *inicializar_algoritmo*, y luego corre el filtro en bucle procesando el dataset sobre las iteraciones indicadas. También invocará a los métodos de clusterizado de la Sección 4.2.1 entre los pasos 4 y 5 del filtro de partículas.
- **Planpruebas.m**: Utilizado para invocar a *ejecución.m* con distintas decisiones de diseño, para así poder probar distintos datasets con distintos parámetros de manera consecutiva, permitiéndo realizar investigaciones paramétricas.

6. Utilidades

Esta carpeta contiene una serie de métodos genéricos que nos serán útiles en momentos específicos de la ejecución.

- **Acotar.m**: Realiza una transformación lineal de una matriz entre el límite inferior y superior, con salida un número entre 0 y 1. Los valores fuera de los límites se transformarán a 0 si queda por debajo del límite inferior y a 1 si queda por encima del límite superior.

- **Aplicar_movimiento.m**: Aplica una traslación y un giro a unas coordenadas.
- **B_bin_exceso.m**: Realiza una búsqueda binaria por exceso. Esto es, devolverá el índice del elemento del array tal que el valor buscado está entre este índice y el anterior.
- **Busqueda_binaria.m**: Implementa el famoso algoritmo de búsqueda en lista ordenada.
- **Galeria.m**: Clase que simula una galería de imágenes. Guarda los títulos de estas, el instante en que fueron tomadas y la carpeta donde se encuentran. El constructor rellena estos campos y la función *getImagen* busca la imagen tomada en el instante pasado por argumento.
- **Get_diff_posyang.m**: Devuelve la diferencia de posición y de ángulo entre dos instantes de tiempo, a partir de la información del lcm. Lo primero que hará será estimar la posición y orientación del coche para cada instante. El problema es que el lcm guarda datos en distintos tiempos, pero no tienen por qué coincidir con el pedido. Para arreglarlo, se buscarán las dos muestras que cubren el intervalo al que pertenece el tiempo solicitado, y se realizará una media ponderada (por cercanía del instante de tiempo a un extremo o al otro) de los valores de posición y orientación, tratando de estimar estos valores en el tiempo requerido. Para calcular la diferencia angular y de posición, bastará realizar las restas respectivas.
- **Max_matrix.m**: Devuelve el máximo valor de una matriz y los índices donde se alcanza.
- **Polar_to_cartesian_grid.m**: Transforma una rejilla polar en una cartesiana, usando el algoritmo descrito en la Sección 3.2.2.3. Los argumentos son la rejilla polar, las medidas y resoluciones del mapa cartesiano de salida y el valor por defecto que se dará a las celdas sobre las que no se tiene información.
- **Representa_celdas_vect.m**: Representa el mapa de las celdas con probabilidad de ocupación por encima de cierto umbral, indicando la velocidad de estas mediante un vector, y un código de colores siendo rojo las celdas dinámicas y azul las estáticas.
- **Representa_clusters_arg.m**: Representa los clusters calculados, indicando su velocidad mediante un código de colores. Blanco indicará cluster estático (con velocidad por debajo de cierto umbral) y el resto de colores indican la dirección de la velocidad.
- **Representa_clusters_vect.m**: Representa los clusters indicando con flechas la velocidad de los mismos (tanto la dirección como el módulo). Los colores están pensados para que en instantes sucesivos, los clusters asociados preserven su color, y así será más visual a la hora de observar las gráficas.
- **Representa_partículas.m**: Representa la posición de las partículas en un eje de coordenadas.
- **Representa_velocidades.m**; Utiliza el mismo código de colores que *representa_clusters_arg* para representar las velocidades de las celdas.
- **Representacion.m**: Representa las gráficas de cada iteración, invocando a los distintos métodos de representación.
- **Transformar_arg_color.m**: Realiza la correspondiente transformación de una velocidad al código de colores correspondiente. Este código viene detallado en el Apéndice F.

7. Fichero extra: decisiones_disenio

Este fichero quizá sea el más importante de todo el software. Contiene una clase que guarda las decisiones de diseño del algoritmo. Así, si se quiere cambiar cualquier parámetro de éste, como por ejemplo el tamaño del área de visión, algún umbral, los factores de DBSCAN o cualquier otro, bastará acudir a este fichero y cambiar el valor correspondiente. Luego, llamar al constructor, que devolverá una instanciación de esta clase contenedora, que será argumento de la gran mayoría de los métodos. A continuación, se da una breve explicación del significado de cada uno de estos parámetros:

- **V**: Número total de partículas.
- **V_b**: Número de partículas recién nacidas a cada paso.
- **P_s**: Probabilidad de supervivencia de una partícula.
- **P_b**: Probabilidad de nacimiento de una partícula en una celda libre.
- **Min_pA**: Probabilidad de asociación mínima de las celdas.
- **Ancho**: Ancho del mapa cartesiano en metros.
- **Largo**: Largo del mapa cartesiano en metros.
- **Lado_celda**: Longitud del lado de una celda en metros.
- **Max_vel**: Máxima velocidad de la vía en metros por segundo.
- **Desviacion_velocidad**: Desviación típica del ruido con distribución normal aplicado en los nacimientos asociados sobre la velocidad de la celda o del cluster.
- **P_quieta**: Probabilidad de que un nacimiento no asociado tenga velocidad nula.
- **Max_pred_certainity**: Máxima masa predicha. Debe fijarse a un valor menor que uno para evitar dividir por cero en algunos casos al combinar distintas masas.
- **Umbral_velocidad**: Umbral de velocidad en metros por segundo por debajo del cual se considera velocidad como nula**.**
- **Desviación_movimiento**: Desviación típica del ruido con distribución normal aplicado al mover una partícula a cada iteración.
- **Num_capas**: Número de capas del láser.
- **Ancho_capa**: Amplitud de cada capa medida en grados.
- **Angulos_inf_capas**: En grados, dirección angular de las capas respecto al coche (midiendo el ángulo inferior de las amplitudes).
- **Altura_lidar**: Altura a la que se está colocado el láser respecto del suelo.
- **Num_max_puntos_angulo**: Número máximo de puntos detectables por cada resolución angular del Lidar.
- **Res_dist**: Resolución del mapa polar inicial en distancia, en metros.
- **Res_ang**: Resolución angular del mapa polar inicial, en radianes.
- **Max_dist**: Distancia máxima considerada por el láser en metros.
- **Min_ang**: Ángulo mínimo considerado por el láser (siendo el ángulo 0 la frontal del coche), en radianes.
- **Max_ang**: Ángulo máximo considerado por el láser (siendo el ángulo 0 la frontal del coche), en radianes.
- **Desv_ocupacion**: Desviación típica de la gaussiana de ocupación.
- **Umbral_densidad_pared**: Densidad mínima de impactos de una capa para considerar que esta está chocando completamente con algo. Usado en la detección de suelo dinámico.

- **Percentil_localización**: Percentil utilizado para localizar paredes o suelos allí dónde se supera *umbral_densidad_pared*.
- **Error_distancia**: Número de celdas máximo de error (podría estar medido también en metros) para considerar que dos impactos totales de distintas capas chocan con la misma cosa (una pared).
- **Nceldas_busqueda_contraste**: Número de celdas a la redonda dónde buscar otros impactos para considerar que un impacto es verdadero o no. Usado en el filtrado de la nube de puntos.
- **Min_verosimilitud**: Verosimilitud mínima por debajo de la cual se considera certeza cero.
- **Max_verosimilitud**: Verosimilitud máxima por encima de la cual se considera verosimilitud máxima.
- **Distancia_clust**: Distancia utilizada para clusterizar en DBSCAN suavizado, en metros.
- **Media_minima_clust**: Umbral de masa media mínima utilizado en DBSCAN suavizado.
- **Despreciar_clust**: Mínimo de celdas de un cluster por debajo del cual se desprecia.
- **Umbral_reajuste**: Mínimo porcentaje de coincidencia a partir del cual la confianza de asociación CA comenzará a tener valor positivo.
- **Maximo_reajuste**: Máxima confianza de asociación.
- **Umbral_distancia**: Máxima distancia de error entre dos clusters para considerar calcular el índice de incidencia IDI.
- **Umbral_representacion_vel**: En las representaciones de celdas ocupadas, masa ocupada mínima para representar una celda como ocupada o su velocidad.
- **Guardar_deteccion**: Flag que al estar activada indica que quiero guardar una carpeta extra con gráficas de la fase de detección.
- **Extra_representacion**: Flag al estar activada indica que quiero guardar información extra de la prueba.
- **Nombre_dataset**: Nombre del dataset a ejecutar.
- **Carpeta_deteccion**: Nombre de la carpeta en la que se guardará información de la detección (si *guardar_deteccion* está activado).

Además, en este fichero se guardan las distribuciones de nacimiento de partículas asociadas y no asociadas, así como la función de difusión de masa libre α.

D Láser utilizado: Lidar

La tecnología utilizada para detectar los impactos es un láser especial denominado LIDAR[9] (*Light Detection and Ranging o Laser Imaging Detection and Ranging*). En concreto, se cuenta con el modelo IBEO LUX HD. Este consta de 4 capas, cubriendo cada una de ellas 0.8°, llegando a cubrir 3.2° en vertical. En cuanto a la amplitud horizontal, todas ellas están plenamente operativas en 85° (entre 35° y -50°) y se tendrán al menos dos operativas en 110° (entre 50° y -60°). Tiene un rango de 90m, y es capaz de detectar tres impactos alineados, ya que puede tomar muestras a través de polvo, gotas de lluvia o cristales. Por último, puede trabajar a frecuencias de 12.5, 25 y 50 Hz, perdiendo resolución y calidad de barrido a mayor frecuencia.

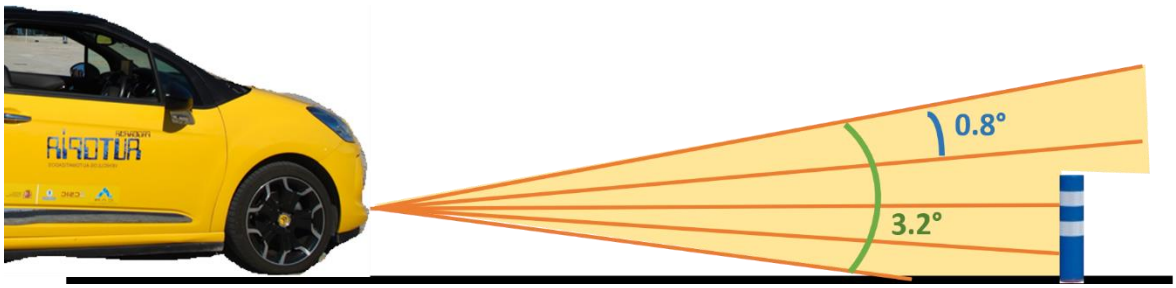

***Figura D-1**: Amplitud vertical del LIDAR*

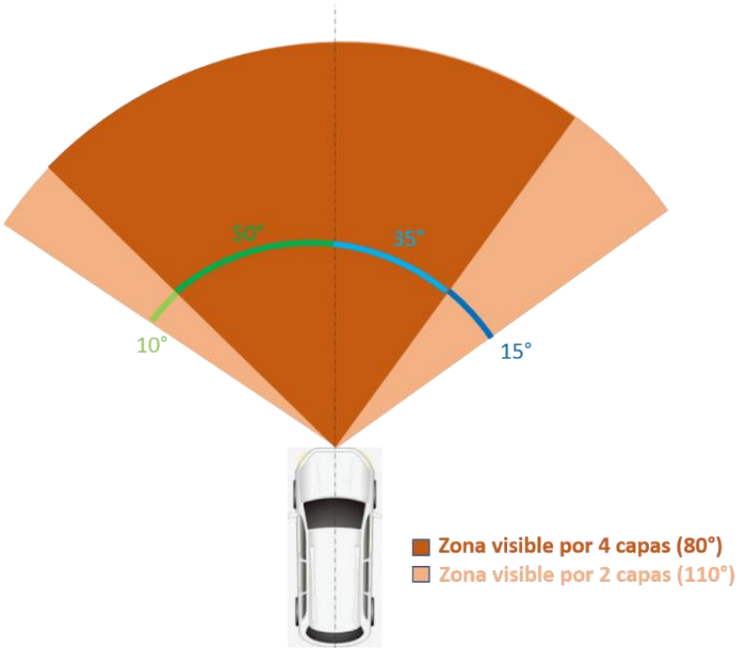

***Figura D-2**: Amplitud horizontal del LIDAR*

El hecho de que tenga solo 4 capas que cubren 3.2° verticales será un poco problemático pues no tiene todo el campo de visión deseable. Sin embargo, será suficiente para los propósitos de este trabajo. Además, toda la implementación está llevada a cabo para

un número variable de sensores (que en este caso son solo 4 capas, pero podrían ser más capas e incluso tecnologías de sensores diferentes, como un radar o una cámara).

E Comentarios sobre transformación de rejilla polar a cartesiana

Los pasos a seguir por el algoritmo son:

1. Representar la rejilla cartesiana subyacente y calcular los puntos medios de las celdas polares.
2. Para cada celda cartesiana, se identifican qué puntos medios de celdas polares pertenecen a su región.
3. Se hace la media de los valores de las celdas polares asociadas a las celdas cartesianas. Se rellenan los valores de las celdas cartesianas con algún punto en su interior.
4. Las celdas cartesianas fuera del rango polar, se fijan a valor desconocido.
5. Para las celdas cartesianas en rango pero sin puntos medios, se infiere su valor calculando la media de los valores adyacentes conocidos.
6. Se ha logrado transformar la rejilla polar a cartesiana mediante aproximaciones.

A continuación, se exponen figuras aclarativas.

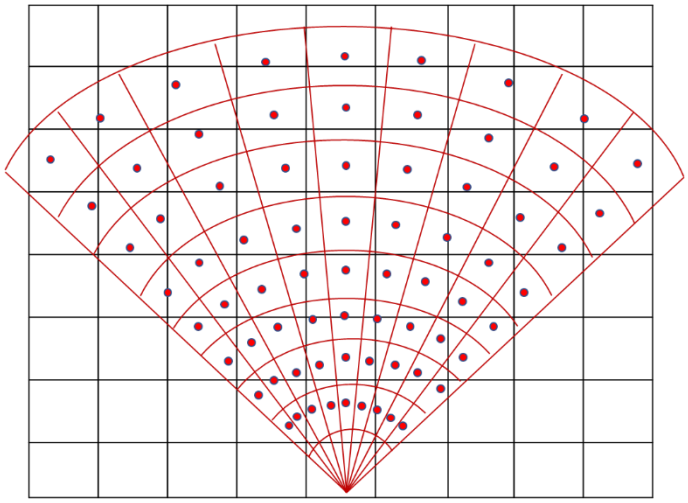

***Figura E-1**: Paso 1 de la transformación de rejilla polar a cartesiana*

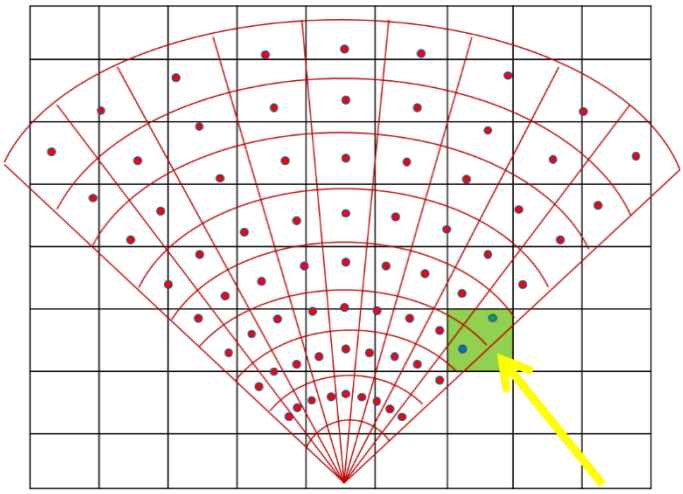

***Figura E-2**: Paso 2 de la transformación de rejilla polar a cartesiana.*

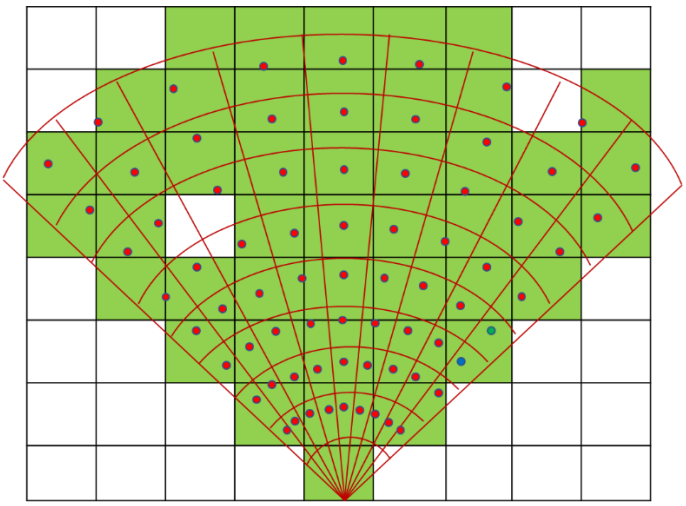

***Figura E-3**: Paso 3 de la transformación de rejilla polar a cartesiana.*

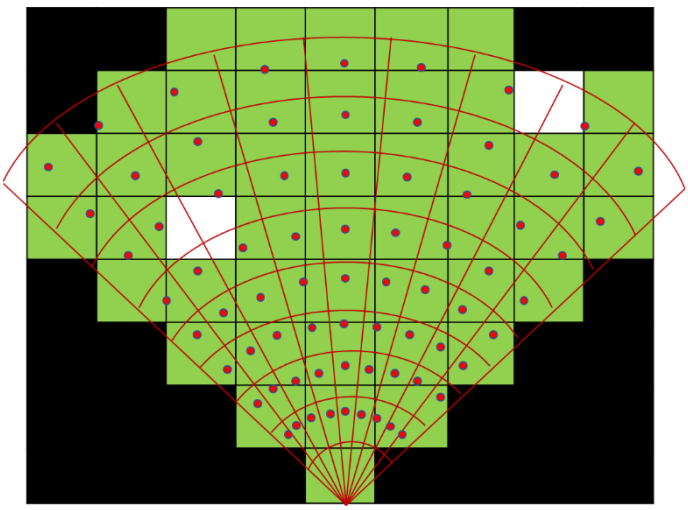

***Figura E-3**: Paso 4 de la transformación de rejilla polar a cartesiana.*

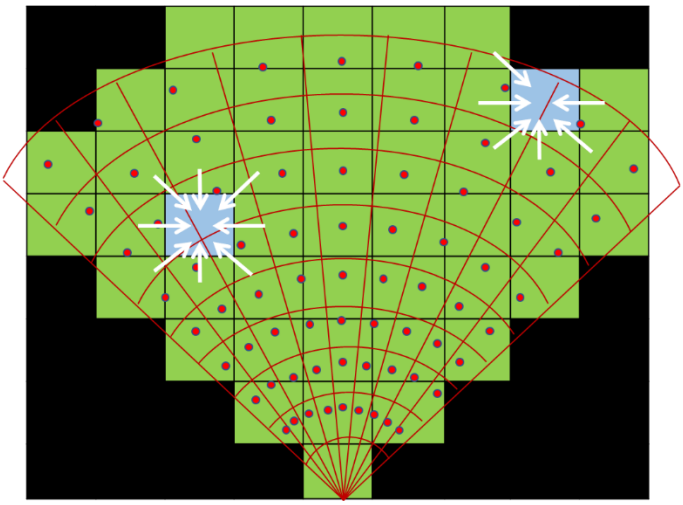

***Figura E-5**: Paso 5 de la transformación de rejilla polar a cartesiana.*

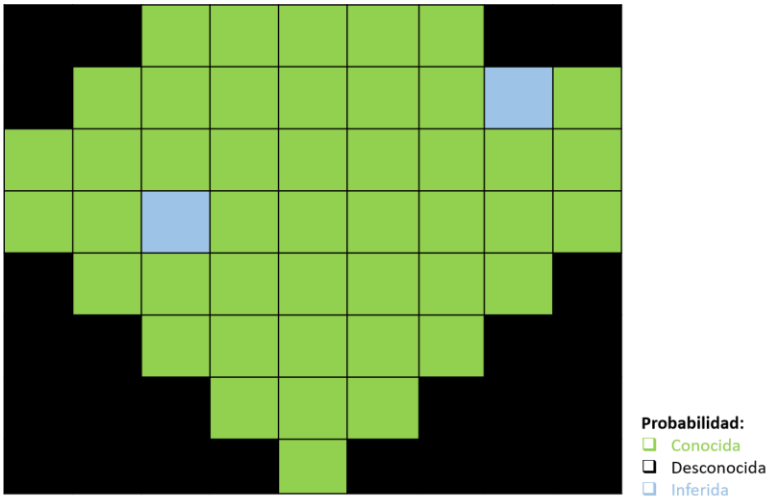

***Figura E-6**: Paso 6 de la transformación de rejilla polar a cartesiana.*

Como ya se ha visto, la aproximación para transformar una rejilla polar a cartesiana, que consiste en rellenar los valores de las celdas cartesianas con las celdas polares cuyo punto medio cae en tales celdas cartesianas, es muy útil, siempre y cuando haya más densidad de celdas polares que cartesianas. De ser así, se garantiza que en la mayoría de los casos se podrá asociar valores a las celdas cartesianas, pues habrá una o más celdas polares con las que hacerlo. Esto sucede a distancias cercanas al láser, pues el tamaño de las celdas polares cerca del origen es menor y por tanto hay mayor densidad de estas. Sin embargo, habrá cierta distancia a partir de la cual el tamaño de las celdas polares sea mayor que el de las celdas cartesianas, quedando demasiadas celdas cartesianas sin valor conocido.

Así, si se trabajase con mapas de ocupación más amplios, esto es, con más rango de visión, quizás la aproximación para transformar la rejilla no sea la mejor, debido al gran número de celdas cartesianas sin valor. Así, podría definirse una distancia a partir de la cual cambiar el paradigma de transformación. A partir de esa distancia, una idea sería calcular la

celda polar donde cae el punto medio de las celdas cartesianas, y rellenar el valor de la celda cartesiana con el valor de la celda polar correspondiente (procedimiento contrario al llevado a cabo para distancias cercanas).

Una posible distancia a partir de la cual cambiar este paradigma podría ser aquella distancia a partir de la cual haya más celdas cartesianas sin valor que con. Esta distancia es aquella donde el área de una celda polar sea el doble de la de una celda cartesiana. Así, fijado el lado l de una rejilla cartesiana, la resolución angular A y la resolución en distancia D de una rejilla polar, así como la distancia d a la que se encuentra la celda, se tiene que el área de cada celda es:

$$A_{pol} = \frac{\pi A}{360}[(\mathrm{d} + \mathrm{D})^2 - d^2] \qquad A_{cart} = l^2$$

Por tanto, la distancia a la cual el área de la celda polar es el doble que la de la celda cartesiana es:

$$d = \frac{360l^2}{\pi AD} - \frac{D}{2}$$

Así, a partir de esta distancia puede que el rendimiento del algoritmo de transformación de rejillas descrito en la Sección 3.2.2.3 decaiga. Si se utilizan los valores, esto es, $A = 0.5°$, $D = l = 0.2m$, se tiene que $d = 35m$ aproximadamente. Luego al menos hasta los 35m nuestro algoritmo funciona de manera óptima.

F Detección de la distancia de impacto de las capas con el suelo

Para calcular correctamente la verosimilitud de inclinación de las celdas, se necesita saber a qué distancia del coche choca cada capa con el suelo. A priori, debería ser fácil, pues basta conocer la altura del Lidar y la inclinación de las capas para calcular esta distancia usando trigonometría básica. Sin embargo, hay que contar con el pitch[14], que hará variar esta distancia constantemente.

Debido a que no se tienen sensores para estimar la inclinación del coche en cada momento, se debe calcular la distancia de impacto mediante software. Este cálculo debe ser dinámico debido al continuo cambio de inclinación del coche. El primer paso para lograrlo es decidir si una capa está impactando completamente con algo (un suelo o una pared).

Para ello, se observa la densidad de impactos de cada capa. Esto es, el número de resoluciones angulares con impacto partido del número de resoluciones angulares. Si esta supera cierto umbral, se decide que la capa está impactando completamente con algo. Este umbral depende de la resolución angular fijada. En la versión final del software se usa 0.5 grados de resolución angular y 0.8 como umbral. Si se decide que se está impactando completamente con algo, se estima que la distancia a la que ocurre es en el percentil 95 de las distancias de los impactos. No se usa el percentil 100, es decir, el impacto más lejano, pues a menudo se encuentran con datos atípicos que se deben a errores del hardware. Además, siempre será más seguro detectar este suelo o pared antes de donde realmente está que después.

Una vez hecho esto, puede se tiene una distancia candidata de impacto de capa con el suelo. Puede ser que realmente no esté impactando con el suelo sino con una pared o un objeto de gran dimensión, así que hay que idear una manera de distinguir estos dos casos. Se decide que la capa impacta con el suelo si y solo si ninguna otra capa está impactando completamente a una distancia cercana a esta. Así, si una capa impacta completamente con algo y ninguna otra capa lo hace cerca, es razonable pensar que está impactando con el suelo, y se fija esta distancia para la verosimilitud de inclinación. Si por contra, más capas están impactando completamente en esta distancia (o bastante cerca, se define un umbral), será porque realmente hay un objeto vertical a esa distancia, y por tanto no es el suelo.

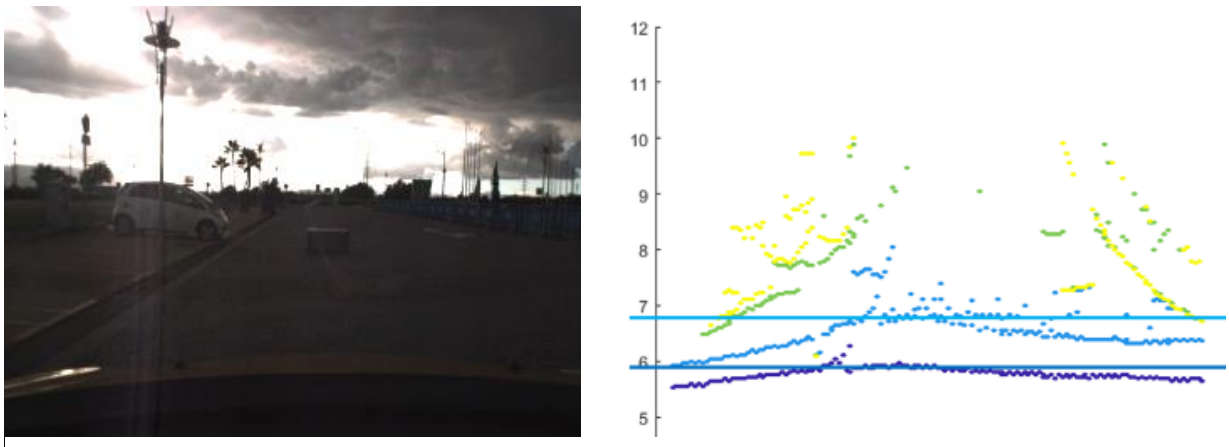

***Figura F-1**: Detectando dinámicamente el impacto de las capas inferiores con el suelo a partir de la nube de puntos (representada en polares)*

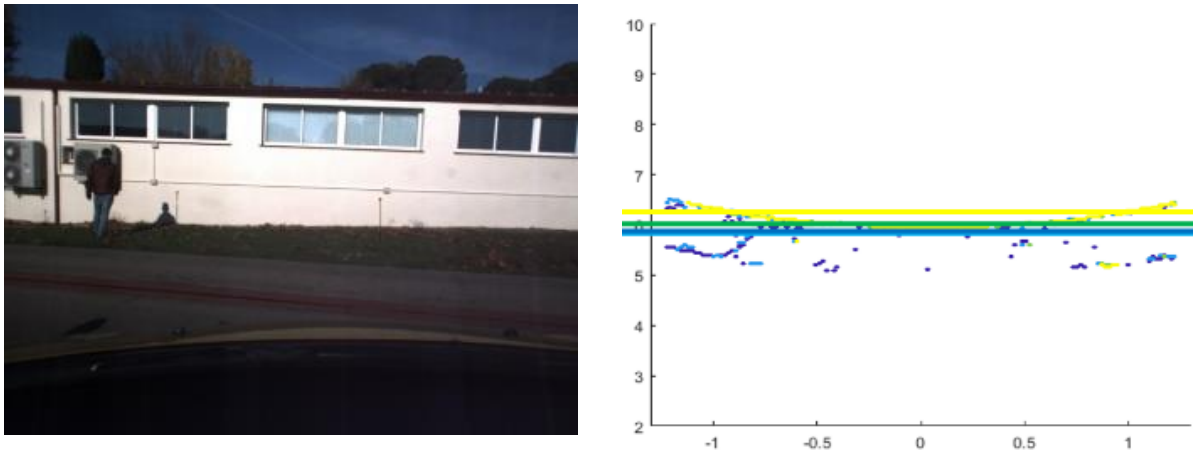

***Figura F-2**: Detectando dinámicamente una pared a partir de la nube de puntos (representada en polares)*

En las imágenes anteriores, se muestra mediante rayas horizontales las distancias candidatas en las que cada capa del Lidar impacta con algo completamente. En el primer ejemplo, se decide que las dos capas inferiores están impactando con el suelo, pues estas distancias de impacto están "alejadas" unas de otras. Así, se fijan estas distancias como la distancia de impacto de la capa contra el suelo, afectando a la verosimilitud de inclinación: los puntos de impacto de esa capa por detrás de la raya no sumarán a la probabilidad de ocupación pues tendrán verosimilitud nula. En el segundo ejemplo, se observa que las cuatro capas están impactando completamente contra algo a una distancia muy cercana entre ellas. Por ello, se decide que esto sí corresponde a un objeto vertical y por tanto la verosimilitud de inclinación se mantiene con sus valores por defecto. Aun así, la verosimilitud por detrás de esta pared también baja a cero debido a la verosimilitud de ocupación (siguiendo una gaussiana), pero al menos no hay una caída lineal de la verosimilitud de inclinación hasta estas distancias.

G Filtrado inicial de la nube de puntos

Aun calculando dinámicamente la distancia de impacto de cada capa con el suelo, se sigue teniendo bastante ruido, debido a que el suelo no siempre es plano y se detectan impactos con éste con verosimilitud no nula. Como se observa en la figura, muchos impactos de la capa inferior siguen aportando probabilidad de ocupación (y, por tanto, ruido) aun cuando estos claramente provienen de impactos con el suelo. Los impactos por debajo de la lineal azul generan ruido.

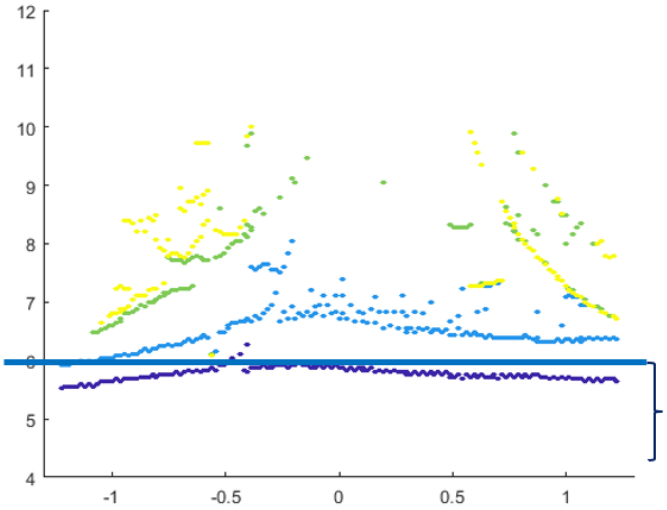

***Figura G-1**: Ruido emitido por el suelo*

Esto no sería problemático si se tuvieran más capas, pues la verosimilitud de estos puntos sería mínima (debido a la caída lineal de la verosimilitud de inclinación) con respecto a los pesos de todas las demás capas. Sin embargo, el ruido que emite este fenómeno es suficiente para que moleste en desarrollos posteriores. Por ello mismo, conviene eliminarlo.

Así, se lleva a cabo un filtrado previo de la nube de puntos. El único objetivo de este filtrado es diferenciar falsos positivos de verdaderos positivos (tratando impactos con el suelo como falsos positivos). Sería deseable no tener que aplicar este filtrado, pues las ecuaciones de probabilidad y verosimilitud de ocupación ya manejan este fenómeno. Sin embargo, el tener cuatro capas (y en algunas áreas solo tres o dos activas) hace que un falso positivo pondere demasiado en el cómputo de la probabilidad de ocupación global. Por ello, conviene arreglarlo añadiendo este filtro extra.

Para llevarlo a cabo, se fija un radio de búsqueda (parámetro del software). Para cada punto de la nube, si no encuentro otros puntos de otras capas en este radio, decidiré que es un falso positivo. Si se tiene evidencia de otras capas de puntos en esa zona, será un verdadero positivo. La única diferencia entre un impacto falso y uno verdadero es que la gaussiana de ocupación queda recortada, evitando que la probabilidad suba por encima de 0.5. Además, la verosimilitud caerá a cero siguiendo una gaussiana sin importar que haya más impactos a continuación.

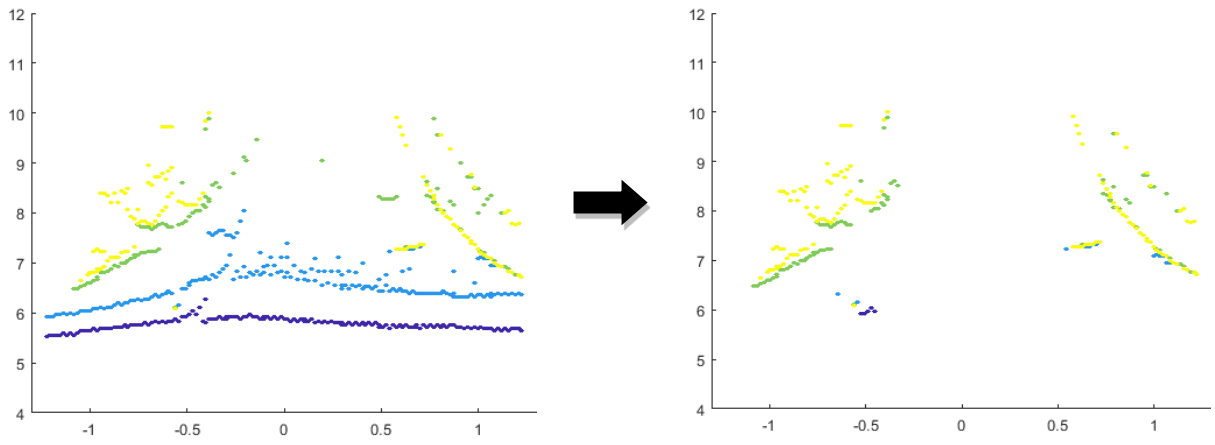

***Figura G-2**: Filtrado de la nube de puntos.*

H Integración de la detección dinámica del suelo y filtrado de la nube de puntos

Las técnicas descritas en los apéndices F y G eliminan notablemente el ruido, especialmente aquel introducido por el impacto de las capas con el suelo. A continuación, se muestra unas figuras ilustrativas. De izquierda a derecha y de arriba abajo: instante capturado, ocupación con aproximación inicial, ocupación con detección del suelo dinámica, y ocupación con detección del suelo dinámica y filtrado de la nube de puntos.

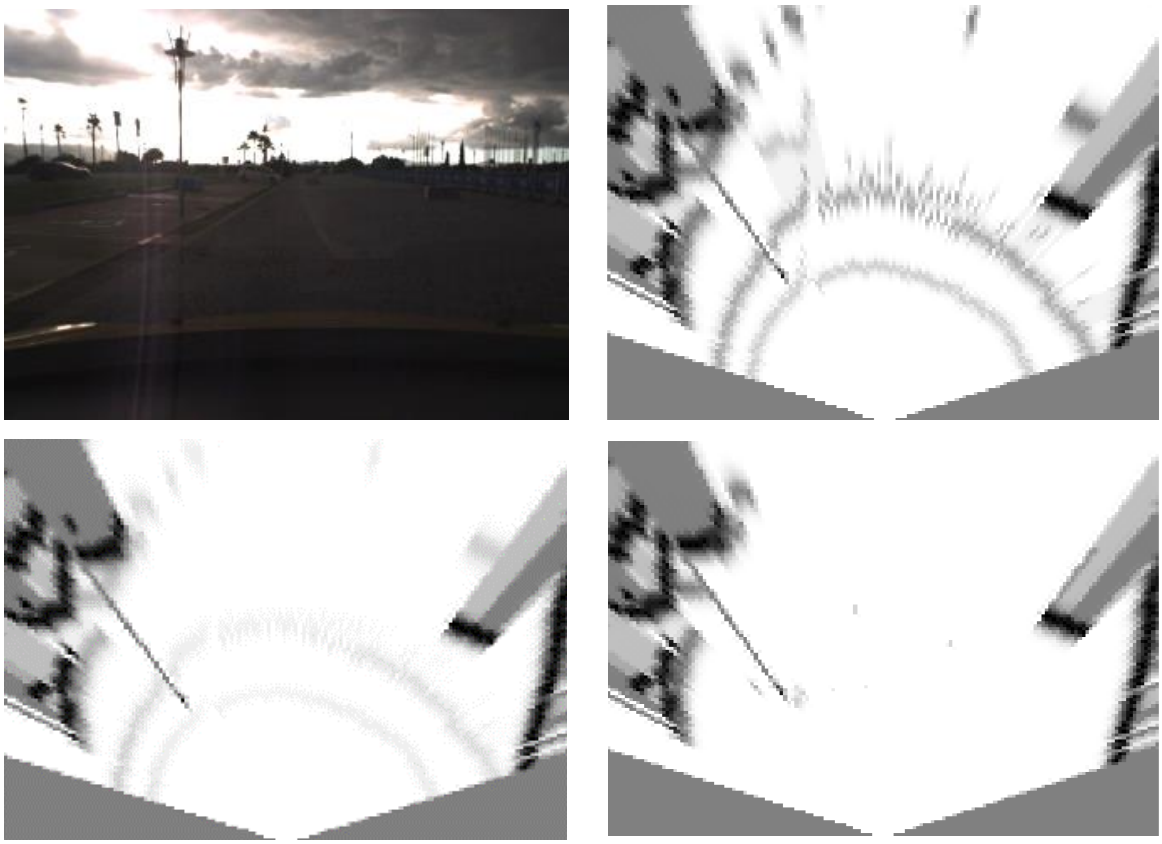

***Figura H-1**: Mejoras aportadas por la detección automática del suelo y el filtrado previo de la nube de puntos.*

I Masa de evidencia

La masa de evidencia es un concepto que permite no solo dar información sobre probabilidades, sino también sobre la certidumbre de estas probabilidades. Es un concepto muy amplio y tiene su utilidad en muchas ramas, pero en este anexo está limitado a dar explicaciones de la aplicación directa al proyecto.

En este caso, la masa de evidencia será una función $\boldsymbol{M}: 2^{\Omega} \longrightarrow [0, 1]$, donde $\Omega = \{O, L\}$. O representa ocupado, y L libre. Así:

- $\boldsymbol{M}(O)$ es la probabilidad de estar ocupado
- $\boldsymbol{M}(L)$ es la probabilidad de estar libre
- $\boldsymbol{M}(OL)$ es la probabilidad de estar libre y ocupado (esto es, simboliza la incertidumbre)
- $\boldsymbol{M}(\emptyset) = 0$

Deber cumplirse por tanto $\boldsymbol{M}(O) + \boldsymbol{M}(L) + \boldsymbol{M}(OL) = 1$

Se observa que, para determinar completamente una masa de evidencia (referida a la ocupación), basta dar los valores $\boldsymbol{M}(O)$ y $\boldsymbol{M}(L)$, pues $\boldsymbol{M}(OL)$ puede ser despejada de la ecuación anterior.

Además, la probabilidad de que una celda esté ocupada será:
$P(O) = \boldsymbol{M}(O) + 0.5 * \boldsymbol{M}(OL)$

La teoría de Dempster-Shafer aporta una manera de combinar dos masas de evidencia. Esto es algo natural que hacer, pues es equivalente a tener dos jueces opinando sobre algo, y ser capaz de englobarlos en un juez superior que tiene en cuenta las opiniones de ambos. Se denotará esta combinación con el operador $\oplus$, y viene dado por las ecuaciones:

$$(m_1 \oplus m_2)(\emptyset) = 0$$

$$(m_1 \oplus m_2)(A) = \frac{1}{1 - \zeta} \sum_{B \cap C = A} m_1(B) m_2(C)$$

Siendo ζ el conflicto, dado por:

$$\zeta = \sum_{B \cap C = \emptyset} m_1(B) m_2(C)$$

Este operador se utilizará para combinar las predicciones para este instante basadas en la ocupación calculada en los instantes anteriores, con la ocupación calculada por lo medido con el Lidar en este instante.

J Mejoras aportadas por el filtro de partículas a la detección instantánea

En este anexo, se adjuntan una serie de gráficas que evidencian las mejoras que aporta el filtro de partículas a la detección instantánea.

En este primer ejemplo, tenemos de izquierda a derecha: instante capturado, probabilidad de ocupación instantánea y probabilidad de ocupación usando el filtro. En este dataset las pelotas ruedan de una persona a otra, así que el espacio detrás de las pelotas tiene prácticamente 0 de probabilidad de ocupación al aplicarle el filtro, debido a que en instantes anteriores se observó esta zona despejada. El ligero tono grisáceo se debe al factor de difusión de masa libre. Así, si las pelotas se parasen en ese punto, a la larga las dos imágenes serían la misma pues la probabilidad de ocupación en zonas inobservadas tiende a 0.5

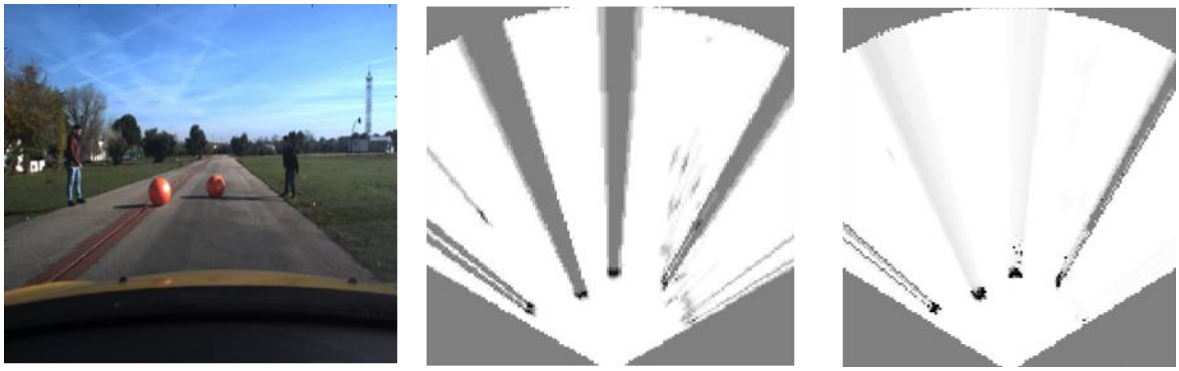

***Figura J-1**: Mejora de detección de ocupación en zonas oclusas.*

En el segundo ejemplo, el coche autónomo está siguiendo al coche de enfrente. Al moverse hacia delante, lugares ya observados pasan a los ángulos muertos inferiores marcados en rojo. Sin embargo, se logra cierta estimación de la ocupación en esta zona gracias al filtro de partículas

***Figura J-2**: Mejora de detección de ocupación en lugares ya observados.*

En la misma situación que en el ejemplo anterior, se representa a la derecha la distribución de las partículas, que se encuentran allí dónde hay masa. Además, se observa más densidad de partículas en los dos coches, pues están en movimiento, mientras en las zonas estáticas (límites de la carretera a los lados) se tienen menos densidad al haberse aglomerado las partículas estáticas.

***Figura J-3**: Distribución de las partículas.*

Una vez más en la misma situación, se representan las velocidades de las celdas. En rojo, las celdas dinámicas y en azul las estáticas. Aunque en general la estimación parece buena, no hay manera de saberlo, ya que no hay velocidades globales, sino a nivel celdilla. Además, hay muchas celdas cuya velocidad es completamente errónea. Esto se arregla en desarrollos posteriores. (Sección 3.4)

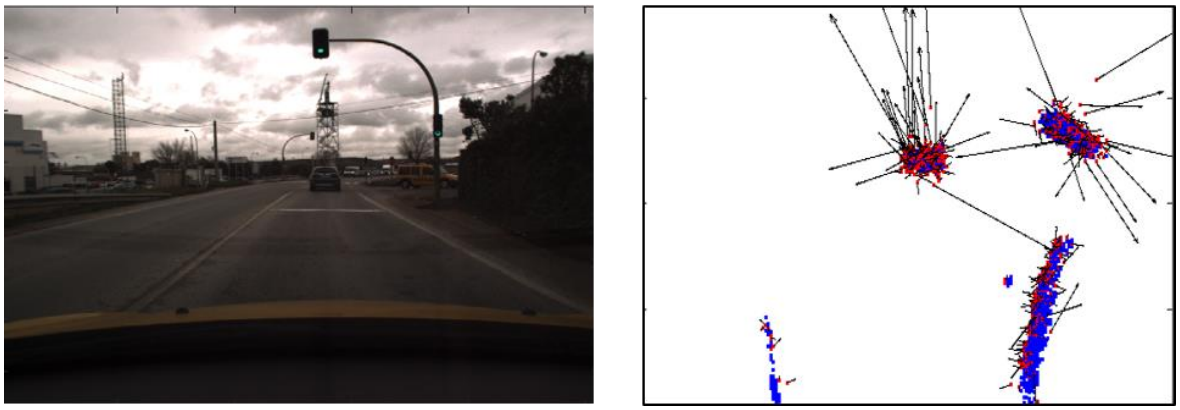

***Figura J-4**: Velocidades de las celdillas.*

K Pseudocódigo de DBSCAN

```
DBSCAN(D, eps, MinPts)
   C = 0
   Para cada punto sin visitar P del dataset D:
      Marco P como visitado
      PtsVecinos = puntosCerca(P, eps)
      Si tamaño(PtsVecinos) < MinPts:
         marco P como RUIDO
      else:
         C = siguiente cluster
         expadirCluster(P, PtsVecinos, C, eps, MinPts)

expadirCluster (P, PtsVecinos, C, eps, MinPts)
   añado P al cluster C
   para cada punto P' in PtsVecinos:
      si P' no está visitado:
         marco P' como visitado
         PtsVecinos’ = puntosCerca(P', eps)
         si tamaño(PtsVecinos’) >= MinPts
            PtsVecinos = PtsVecinos unión con PtsVecinos’
      si P' no es miembro de ningún cluster
         añadir P' al cluster C

puntosCerca(P, eps)
   devolver todos los puntos a distancia menor que eps de P (P
incluido)
```

L Pseudocódigo de DBSCAN suavizado

```
    DBSCAN_suavizado(D, eps, MinOc)
       Clust = 0
       D’ = celdas Cell de D con Cell.masaOcupación > MinOc/3
       Para cada celda sin visitar Cell del dataset D’:
          Marco Cell como visitado
          CeldasVecinas = celdasCerca(Cell, eps)
          Si media(CeldasVecinas.masaOcupación) ≥ MinOc:
             Clust = siguiente cluster
             expadirCluster(Cell, CeldasVecinas, Clust, eps, MinOc)

    expadirCluster (Cell, CeldasVecinas, Clust, eps, MinOc)
       añado Cell al cluster CLust
       para cada celda Cell' in CeldasVecinas:
          si Cell' no está visitado:
             marco Cell' como visitado
             CeldasVecinas’ = celdasCerca(Cell', eps)
             si media(CeldasVecinas'.masaOcupación) >= MinOc
                CeldasVecinas = CeldasVecinas unión con
CeldasVecinas'
          si Cell' no es miembro de ningún cluster
             añadir Cell' al cluster Clust

    celdasCerca(Cell, eps)
       devolver todas las celdas a distancia menor que eps de Cell
(Cell incluido)
```

M Algunos resultados de DBSCAN suavizado

En este anexo se muestran un par de instantáneas del resultado del DBSCAN suavizado, mostrando algunas de sus ventajas.

En el primer ejemplo, tenemos, de izquierda a derecha, imagen de la cámara, masa de ocupación y clustering. Se observa que, aunque la hierba introduce ruido en la zona derecha, el algoritmo de clusterizado es capaz de eliminarlo.

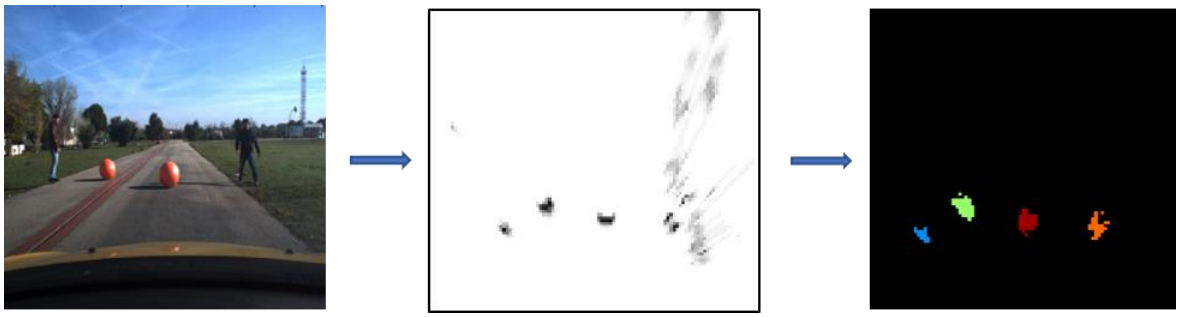

***Figura M-1**: El clustering puede eliminar ruido.*

En este caso, mostramos la imagen de la cámara, masa de ocupación y clustering. El clustering ayuda al cálculo de velocidades de objetos. Se tienen laterales de la vía estáticos, y un coche que se mueve hacia delante, como se aprecia en la imagen de la izquierda.

***Figura M-2**: El clustering ayuda a calcular velocidades de los objetos.*

N Casuística cubierta por los datasets de prueba

Los datasets utilizados incluyen una amplia gama de escenarios:

- Coche estático
- Coche en movimiento
    - En carretera recta
    - En curva
    - En rotonda
- Obstáculos estáticos
    - Conos
    - Quitamiendos
    - Árboles
- Obstáculos en movimiento
    - Objetos del recinto de pruebas (cajas grandes, pelotas, etc)
    - Peatones
    - Otros coches
        - En mismo carril
        - En carril contrario
        - En una rotonda
        - Cruzándose

Sin embargo, estos datasets no cubren por completo todos los escenarios posibles que existen en carretera. Se han recogido los más comunes.

O Ajuste paramétrico: un ejemplo

A continuación se expone un ejemplo sencillo de variación de un parámetro que está bastante incorrelado con el resto: la resolución angular. Aunque parezca un parámetro poco importante, en realidad es fundamental para el correcto funcionamiento del software. El código que reside en el fichero *planpruebas_resolucionAngular* es el que se usó para ajustar este parámetro. Los datasets son: *otro_coche_acelera*, en el que el coche autónomo sigue a otro que acelera en línea recta; *personas*, que consiste en una persona caminando delante de una pared; y *simple*, que consiste en una caja estática y otra con movimiento uniforme con el coche estático. A continuación, se presenta una tabla con las valoraciones de los distintos datasets para las distintas resoluciones angulares. Por último, se da una nota subjetiva a la ejecución, cuantificando cómo de bien lo ha resuelto el software.

Dataset/ resolución angular	OTRO COCHE	SIMPLE	PERSONA
$\frac{1}{8}°$	Apenas coinciden los impactos en las distintas capas por tener tan baja resolución. Probabilidad de ocupación prácticamente nula en todas las celdas. **0**	No se detecta ni la pared por tener impactos tan distribuidos en las muchas celdas. **0**	No se detecta ni la pared por tener impactos tan distribuidos en las muchas celdas. **0**
$\frac{2}{8}°$	Se consigue medir ocupación, pero a menudo los clusters quedan cortados debido a que se tienen agujeros en los objetos debido a que hay más resolución angular que densidad de impactos. **2**	Sin duda el mejor dataset para esta resolución. Se detectan todos los objetos, aunque en ocasiones falla el clusterizado y un objeto se clusteriza como dos. **4**	Se detecta la pared aunque se clusteriza en distintos objetos. Por otro lado, la persona no se llega a detectar por ocupar poco espacio (su planta). **1.5**
$\frac{3}{8}°$	Algunos agujeros en la ocupación y algún cluster (quitamiedos) que se separa en dos en ocasiones. Aun así, bastante buena ejecución. **8.5**	Se detectan todos los objetos y se calculan correctamente los clusters y las velocidades. Sin embargo, se sigue teniendo algunas rendijas que separan los clusters. **8**	Buena detección de la pared, clusterizandola como un solo objeto. Sin embargo, la persona en ocasiones desaparece debido a su pequeño tamaño y que la densidad no es tan grande debido a algún agujero en la detección. Además, detecta espacio libre tras la pared en las resoluciones en las que no hay impactos, cosa que claramente no debería ocurrir. **5**
$\frac{4}{8}°$	Ejecución perfecta, tanto en ocupación como velocidad y clusterizado. No es un 10 pues se tiene un poco de ruido. **9.5**	Muy buena ejecución también, sin errores. **10**	Ejecución perfecta, clusterizando correctamente la pared, con certidumbre cero más allá de ella, y sin perder el rastro a la persona. **9.5**
$\frac{5}{8}°$	Muy buen funcionamiento, tanto en probabilidad de ocupación, como en velocidad y clusterizado. Casi insuperable. **9.5**	Funcionamiento óptimo en todos los sentidos. **10**	Igual que el anterior. Plena detección de la pared en un solo cluster. No se pierde la detección de la persona en ningún instante. **9.5**
$\frac{6}{8}°$	Ejecución casi perfecta también. Celdas más difusas. **9**	Funcionamiento muy bueno en todos los sentidos. Las celdas se difuminan levemente. **9**	Muy buen funcionamiento, pero las celdas vuelven a difuminarse. **9**

***Figura O-1**: Ajuste paramétrico*

De la tabla se observa que las mejores resoluciones son $\frac{4}{8}° = 0.5°$ y $\frac{5}{8}° = 0.625$. No se sigue ejecutando con valores más altos pues las ejecuciones no varían cualitativamente (sí cuantitativamente, cada vez hay menos precisión). Aunque a priori se pensaba que a menor

resolución mayor precisión, esto no es así, como se comprueba, ya que se tienen errores debido a que la densidad de impactos devuelta por el Lidar no es suficiente. Así, se fija la resolución angular a 0.5° (mejor que a 0.625° pues así hay más nivel de detalle cuando se detecten objetos pequeños).

Este proceso se repitió con varios atributos, aunque solo incluyo este como ejemplo, ya que el objetivo de este TFG no es para nada el ajuste paramétrico.

P Código de colores para la velocidad

En ocasiones, a la hora de representar velocidades, interesa más su dirección que su módulo. Un código de colores permite representar fácilmente esta dirección sin añadir flechas que pueden emborronar una imagen. Se utiliza el siguiente código de colores, que marca la dirección del movimiento. Si el módulo de la velocidad es inferior a cierto valor umbral, represento el cluster o la celda de color blanco indicando que lo considero estático.

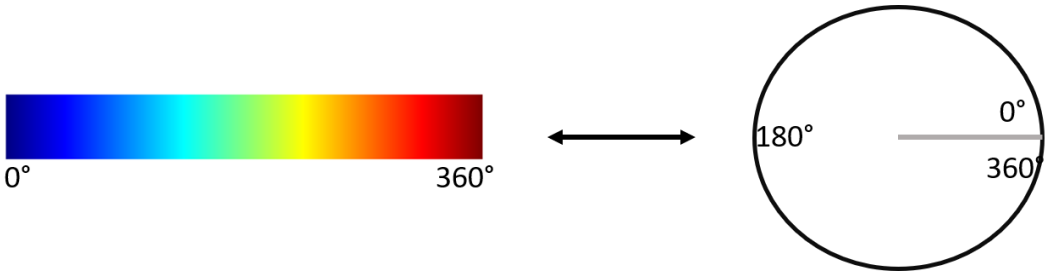

***Figura P-1**: Código de colores para representarla dirección de las velocidades cuyo módulo supera cierto umbral.*